UNIVERSITÉ JOSEPH FOURIER - GRENOBLE I

THESE

Pour obtenir le grade de
DOCTEUR DE L'UNIVERSITÉ JOSEPH FOURIER
Spécialité Mathématique Informatique

par Jacques COMBAZ

Conception de Systèmes Adaptatifs Sûrs et Optimaux

Préparée au sein du laboratoire Verimag et de l'entreprise STMicroelectronics
Soutenue publiquement le 15 Juin 2006

Jury :

Petru	ELES	Rapporteur
François	BODIN	Rapporteur
Jean-François	MÉHAUT	Examinateur
Joseph	SIFAKIS	Directeur de thèse
Jean-Claude	FERNANDEZ	Co-directeur de thèse
Thierry	LEPLEY	Tuteur technique

TABLE DES MATIÈRES

CHAPITRE 1

Introduction

1.1 Motivations

Le contexte global dans lequel s'inscrit ce travail de thèse est celui des applications multimédia embarquées. Ce type d'application est caractérisé par deux exigences qui sont par nature contradictoires. Tout d'abord, les applications multimédia réalisent des traitements algorithmiques très lourds, en terme de consommation mémoire et surtout de puissance de calcul. Pour obtenir une implémentation efficace dans le contexte embarqué (en particulier limité en ressources), il est nécessaire d'utiliser des plates-formes complexes [43] (accélérations matérielles, architectures parallèles, caches, etc...), ainsi que des algorithmes intelligents capables de s'adapter en fonction des données d'entrée. Ceci conduit à une très faible prédictabilité du comportement de l'application, notamment vis-à-vis des durées d'exécution. La seconde particularité des applications multimédia concerne leur caractère temps-réel. En effet, le traitement des flux multimédia impose un tempo qui doit être respecté afin de conserver une qualité de service satisfaisante [26, 14]. Réaliser des systèmes qui respectent des contraintes temps-réel lorsque l'incertitude sur les durées d'exécution est grande et que le système se doit d'être efficace constitue un réel challenge.

Dans le monde industriel, la pratique actuelle est de partir d'un logiciel générique existant, et de l'adapter à la main à la plate-forme d'exécution. En général, le code applicatif spécifique et la plate-forme sont développés en parallèle. Les problèmes liés au temps-réel sont donc traités de manière ad hoc, de sorte que la majeure partie du temps de développement est consacrée à la validation et au débogage.

La solution que nous proposons est basée sur une adaptation dynamique de l'application. Celle-ci s'appuie sur un modèle de type flot de donnée et sur l'observation en temps-réel de l'état du système. Il s'agit d'une approche outil dans laquelle nous cherchons à rendre le code applicatif le plus générique possible. La bonne adéquation entre le code applicatif et la plate-forme, ainsi que la gestion des contraintes temps-réel, ne sont plus réalisés à la main mais sont confiés à un ensemble d'outils. L'intérêt de ce travail est double. Non seulement il permet de concevoir des applications plus robustes, en abordant les problèmes de temps directement au cœur de notre méthodologie, mais il permet également de développer des applications dont le code est réutilisable sur de nombreuses plates-formes.

1.1.1 Applications multimédia

Les progrès du matériel électronique et informatique de ces quinze dernières années permettent aujourd'hui de réaliser des traitements de plus en plus complexes, sur des quantités d'informations ne faisant que s'accroîtrent. Ces progrès technologiques ont permis l'essor des applications multimédia, c'est-à-dire des applications utilisées pour stocker, générer, modifier, transmettre ou encore restituer différents médias simultanément. Les médias concernés peuvent être aussi bien des images fixes, du son, de la vidéo ou encore une interface homme/machine interactive. Les techniques de stockage et de transmission ont elles aussi connu des progrès considérables à la fin des années 90, avec le développement de la compression dédiée à l'image, le son et la vidéo. Ces techniques assurent un compromis idéal entre la qualité d'enregistrement du média et la quantité d'informations — nombre de bits — qu'il requiert pour son stockage ou sa transmission. Cependant, cette optimisation de l'information se fait au prix de traitements informatiques très lourds, tant pour l'encodage que pour la modification ou encore la restitution du média compressé. L'encodage d'un flux multimédia a une durée d'exécution très variable en fonction des données. Ceci est due à l'utilisation d'algorithmes intelligents, capables d'assurer un compromis idéal entre la qualité de l'encodage et sa durée d'exécution. L'utilisation d'heuristiques évitent en effet de tester exhaustivement toutes les possibilités d'encodage. Cette adaptation de l'application aux données d'entrée permet de réduire la durées d'exécution moyenne, mais a tendance à rendre l'application moins prédictible.

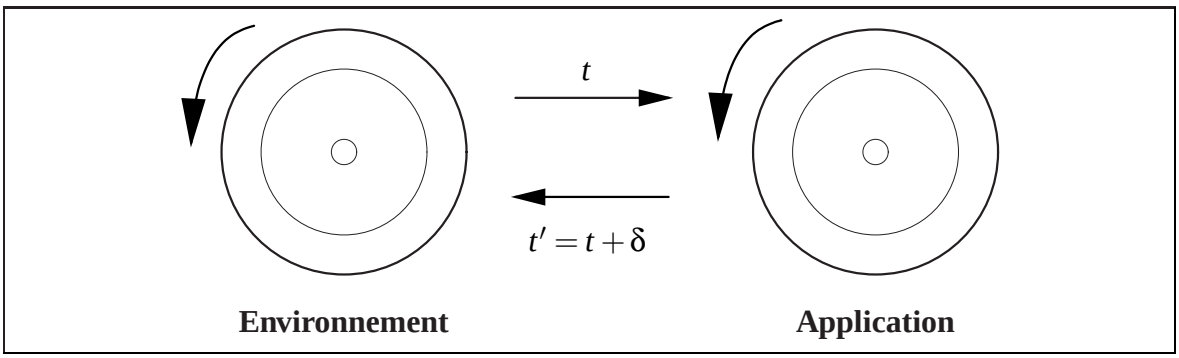


FIG. 1.1: Interaction entre l'application et son environnement d'exécution.

La seconde caractéristique des traitements multimédia concerne leur aspect temps-réel. Une application est dite temps-réel si la validité de sa réponse face à un stimulus dépend non seulement du contenu de cette réponse, mais aussi de la date à laquelle elle est délivrée. Pour prendre un exemple volontairement caricatural, un indicateur d'altitude dans un avion ne peut pas être considéré comme correct s'il existe un laps de temps d'une minute entre la mesure de l'altitude et son affichage, même si l'altitude est mesurée avec une très grande précision. Une application est dite temps-réel si sa réactivité face à son environnement d'exécution vérifie certaines propriétés. Ces propriétés caractérisent la bonne adéquation entre la vitesse de l'application et celle de l'environnement. Ce dernier inclut aussi bien l'utilisateur que tout autre système en interaction avec l'application. Les exigences temps-réel peuvent prendre des formes très diverses. Supposons qu'un stimulus soit produit par l'environnement à la date t (voir figure 1.1), et que la réponse du système à ce stimulus soit délivré à la date $t' = t + \delta$, avec $\delta \geq 0$. Une exigence temps-réel de la forme $D \geq \delta$, où $D \geq 0$ est une constante, est appelée *échéance*. Il s'agit du type de contrainte le plus utilisé dans le domaine du temps-réel.

Concernant les applications multimédia, les contraintes temps-réel proviennent aussi bien des exigences de réactivité par rapport à l'utilisateur, que des flux multimédias eux-mêmes, comme le son ou la vidéo qui sont par nature temporisés. En effet, une séquence vidéo comportant des problèmes de synchronisation entre le son ou l'image, ou encore des variations dans sa vitesse de restitution, ne permet pas de restituer fidèlement la réalité, et peut être très désagréable à regarder.

1.1.2 Contexte : systèmes embarqués

Les systèmes embarqués sont des systèmes conçus spécifiquement pour une application particulière. Les parties matérielles et logicielles sont en général cachées de l'utilisateur dont les interactions avec le système se font via une interface dédiée. Les ressources d'un système embarqué — mémoire de masse, mémoire vive, unités de calcul ou batteries — sont donc fixes, et le plus souvent assez limitées, à la différence d'un ordinateur de bureau, par exemple, dont le matériel évolue en fonction des besoins (rajout de mémoire vive, changement de carte graphique, etc...). Les contraintes du monde embarqué sont plus ou moins fortes en fonction du contexte d'utilisation. Il ne sera évidemment pas possible d'utiliser le même matériel pour un téléphone portable et pour un appareil électroménager, dans la mesure où le premier se doit de pouvoir être autonome d'un point de vue électrique, c'est-à-dire alimenté par batterie rechargeable, alors que le second est alimenté par le secteur.

Aujourd'hui, les systèmes embarqués sont présents partout, aussi bien dans les domaines industriels comme l'avionique et le monde médical, que dans les domaines grand public avec le développement des baladeurs multimédia ou encore de la téléphonie mobile. En 1998, 98% des microprocesseurs vendus dans le monde étaient déjà embarqués. Par la suite, ce taux a continué à s'accroître de telle sorte qu'il approchait les 100% en 2001 [46]. Le marché actuel est très concurrentiel. À cela nous voyons trois raisons principales :

- l'essor de ces systèmes est relativement récent, surtout dans le domaine grand public ; de ce fait, les innovations technologiques sont encore nombreuses et rapides ;
- puisque ces systèmes ne peuvent pas évoluer d'un point de vue matériel, il est nécessaire de développer de nouveaux modèles en fonction des avancées technologiques ; il s'agit donc d'un

marché en perpétuel renouvellement ;
– l'omniprésence des systèmes embarqués dans les produits technologiques grand public traduit un marché de masse, donc particulièrement convoité par les industriels.

Les fonctionnalités assurées par les systèmes embarqués sont toujours plus nombreuses, et toujours plus complexes, à l'image du phénomène actuel de convergence [23]. Cette évolution n'est possible qu'au prix d'un matériel et d'un logiciel, eux aussi, toujours plus complexes. Par exemple, les architectures matérielles utilisées dans l'embarqué sont souvent parallèles et utilisent bon nombre d'accélérateurs matériels ou de processeurs dédiés (comme les DSP), ou encore des caches mémoire. Le logiciel est souvent basé sur un code générique que l'on peut trouver sur les ordinateurs personnels, plus ou moins modifié pour l'adapter aux contraintes de l'embarqué et aux architectures matérielles embarquées très spécifiques.

Les limitations en ressources propres aux systèmes embarqués font que les notions d'efficacité et de rendement se doivent d'être au cœur de leur conception. Cela concerne aussi bien la puissance de calcul, la mémoire ou encore la consommation électrique. La concurrence accrue et l'innovation technologique permanente dans le domaine des systèmes embarqués grand public induisent des temps de mise sur le marché de plus en plus courts. Concevoir des systèmes efficaces et innovants dans un laps de temps très court constitue un réel défi pour les ingénieurs. La pratique actuelle est de partir d'un code générique existant et de l'adapter à la plate-forme cible, qui est très souvent développée conjointement. Pour ce faire, le code applicatif est découpé selon les unités parallèles de la plate-forme, certains morceaux de code sont remplacés par des appels aux accélérateurs matériels, et les algorithmes sont modifiés pour être adaptés à la puissance de calcul de la plate-forme. Nous voyons deux inconvénients majeurs dans cette façon de faire. Tout d'abord, traiter les problèmes de temps-réel à la main est une très grande source d'erreur. De ce fait, il n'est pas possible de construire des systèmes robustes en un temps raisonnable. Aujourd'hui, près de 80% du temps de développement d'une application embarquée temps-réel est utilisé pour la validation et le déboggage [42]. La plupart des erreurs ne proviennent pas d'une mauvaise implémentation d'un algorithme ou une fonctionnalité particulière, mais plutôt des communications et synchronisations entre les différentes tâches qui sont exécutées en parallèle. Ce type d'erreur est directement dépendant des durées d'exécution réelles de l'application. En particulier, une simulation imprécise quant aux durées d'exécution peut cacher des erreurs qui n'apparaissent que sur le produit final. De plus, il est nécessaire de reprendre complètement le code applicatif à chaque fois qu'une nouvelle plate-forme d'exécution est développée. Ainsi, le code applicatif est peu réutilisable, ce qui est très pénalisant pour le temps de mise sur le marché et le coût de développement, surtout dans un contexte de renouvellement permanent des technologies.

Le développement de méthodologies et d'outils adaptés à la conception des systèmes embarqués est devenu une nécessité qui s'impose aujourd'hui aux industriels [23]. Par exemple, le développement des systèmes actuels passe non seulement par la conception de processeurs très performants, mais aussi par l'utilisation de compilateurs super-optimisants, seuls capables de générer du code machine efficace et correct, en un temps et un coût raisonnable. La réalisation elle-même de technologies de compilation reciblables, comme par exemple la technologie de compilation FlexCC2 [12, 19] développée au sein de STMicroelectronics, permet de réduire les temps de mise sur le marché lorsqu'un nouveau processeur est développé.

1.1.3 Vers des systèmes adaptatifs

Les architectures utilisées aujourd'hui dans l'embarqué sont peu prévisibles. Il est par exemple très difficile de prévoir la durée d'exécution d'un bloc de code lorsqu'un cache est utilisé pour accélérer les accès du processeur à la mémoire. De même, la durée d'exécution d'un algorithme très dépendant des données d'entrée ne pourra pas être déterminé statiquement avec précision. D'autres sources de non prédictabilité peuvent aussi provenir de l'environnement d'exécution : interactions avec l'utilisateur ou avec d'autres systèmes. Puisque la validité d'une application temps-réel dépend des durées d'exécutions, il est très difficile dans ce contexte de concevoir des systèmes corrects. Actuellement, deux écoles s'opposent quant à la gestion des contraintes temps-réel.

Temps-réel critique. Dans l'approche temps-réel critique, appelée parfois "temps-réel dur", on cherche à garantir que le système respecte les contraintes temps-réel données. Ces approches sont utilisées à chaque fois que la violation d'une contrainte de temps peut remettre en cause l'intégrité du système, et plus particulièrement dès qu'une notion de sécurité est en jeu. Par exemple, l'impact d'une erreur dans un ordinateur personnel n'est pas du tout le même que celui du dysfonctionnement du système de navigation d'un avion. Les approches temps-réel critique sont utilisées dans l'avionique ou encore dans la conceptions des centrales nucléaires.

Pour garantir le respect des contraintes temps-réel, il est nécessaire de mettre en œuvre des analyses de type pire-cas. Ces dernières sont basées sur des approximations conservatives de la dynamique du système, et sur une allocation statique des ressources. Ces analyses conduisent le plus souvent à surdimensionner les ressources nécessaires à l'exécution du système, que ce soit en terme de puissance de calcul, de consommation mémoire et électrique. C'est le prix à payer pour assurer que le système fonctionnera dans tous les cas de figure, ce qui est une nécessité absolue en ce qui concerne les systèmes temps-réel critiques.

Temps-réel souple. A la différence du temps-réel critique, les approches temps-réel souple (ou mou) considèrent que les contraintes temps-réel peuvent être violées. Ces approches sont utilisées lorsque la violation des contraintes temps-réel ne remet pas en cause l'intégrité du système, ou encore lorsque certains dysfonctionnements du système sont tolérés étant donné qu'il s'agit d'un système non critique, ne remettant pas en cause la sécurité d'autrui. On rencontre de tels systèmes dans les télécommunications ou encore le multimédia. Même s'il serait préférable de garantir le bon fonctionnement du système dans tous les cas de figure possible, la mise en place d'une approche temps-réel critique serait trop pénalisante pour les performances et/ou le coût du système. Ainsi, les approches temps-réel souple s'intéressent le plus souvent à l'optimisation de certains critères tels que la consommation mémoire et électrique, ou encore la vitesse d'exécution. Ces approches tentent de réaliser un compromis idéal entre le taux de contraintes temps-réel violées et l'optimisation des critères de performance du système.

Cependant, cette approche nécessite souvent des mécanismes additionnels capables de limiter le taux de contraintes violées (par exemple, en utilisant un tampons mémoire) et/ou éviter que le système ne se bloque suite à une erreur. Ces mécanismes peuvent être coûteux, en particulier dans le domaine du multimédia dans lequel de très grande quantité d'information sont manipulées. De plus, les systèmes obtenus par une telle approche sont souvent moins robustes que

des systèmes conçus directement pour garantir les contraintes.

Les deux approches temps-réel critique et temps-réel souple sont par nature opposées. Elles concernent deux communautés différentes, qui utilisent des méthodologies, des outils et des plates-formes en général différents. Il est souvent admis que cette séparation reste inévitable, puisque l'utilisation d'analyses pire-cas pour assurer les contraintes temps-réel critiques limite inévitablement l'efficacité de l'application lorsque l'incertitude sur le système et/ou l'environnement est grande. Cette séparation tend à s'accentuer, en particulier pour les systèmes embarqués grand public, pour deux raisons principales :

– L'absence de techniques efficaces d'analyse des comportements pire-cas, tant en terme de durée d'exécution, de consommation mémoire que de consommation électrique.
– La différence entre le comportement moyen et le comportement pire-cas ne fait que s'accroître du fait de l'utilisation de logiciel mais aussi de matériel sophistiqué (technologies de cache, spéculation, ou de *pipeline*). Le matériel récent permet certes de réduire de façon très importante la durée d'exécution moyenne d'une application, mais tend à augmenter la durée d'exécution pire-cas, ou du moins à ne pas réduire autant la durée d'exécution moyenne et pire-cas [39]. La présence d'environnements peu prévisibles ne fait que renforcer la différence qui existe entre les analyses basées sur le cas moyen et celles qui sont basées sur le pire-cas.

Pour arriver à concevoir des systèmes à la fois sûrs et performants, il est essentiel de développer des techniques basées non pas sur des choix statiques, mais au contraire sur une adaptation dynamique de l'application à la plate-forme et à son environnement d'exécution, en fonction de son comportement réel. Une telle adaptation ne peut être réalisée que si l'application contient suffisamment de libertés de contrôle. Les algorithmes de traitement multimédia, et plus particulièrement d'encodage de flux multimédia, sont généralement des heuristiques sur lesquelles il est possible d'intervenir sans altérer leur fonctionnalité. Les approches temps-réel souple existantes sont généralement basées sur des techniques de contrôle. Cependant, le contrôle est réalisé à gros grain sur des paramètres globaux comme le teux d'utilisation du processeur ou encore le nombre d'échéance ratées. Un tel contrôle ne permet pas d'assurer le respect des contraintes temps-réel.

1.2 Etat de l'art

Dans tout développement de logiciel il existe des choix qui ne font pas forcément partie de sa spécification. Ces choix concernent aussi bien l'affectation des tâches ou opérations aux unités de calcul d'une architecture matérielle, que l'ordre d'exécution de ces opérations ou encore le "degré de précision" de ces calculs. De nombreuses techniques ont été développées afin de résoudre ces différentes formes d'indéterminisme, avec en général pour objectif d'imposer certaines propriétés au système, ou encore d'en maximiser les performances. En général, on parle d'ordonnancement pour les techniques qui abordent les problèmes relatifs à l'ordre d'exécution, et on parle de contrôle lorsqu'il s'agit de techniques d'adaptation dynamique des autres paramètres du système.

1.2.1 Techniques d'ordonnancement statique simples

On parle d'ordonnancement statique lorsque les choix d'ordonnancement sont faits avant l'exécution de l'application, et ne sont pas remis en cause durant cette exécution. Les techniques d'ordonnancement statique sont donc efficaces pour des problèmes simples dont tous les paramètres sont connus à l'avance : l'ensemble des tâches à ordonnancer, leurs caractéristiques ainsi que celles de plate-forme d'exécution sont connus. Ces techniques ont été développées dans les disciplines de la recherche opérationnelle et de l'informatique.

Les problèmes d'ordonnancement abordés varient en fonction des applications visées. L'industrie s'est par exemple beaucoup intéressée à l'ordonnancement de tâches sur plusieurs machines, à travers les techniques de recherche opérationnelle. En effet, la plupart des produits issus de l'industrie nécessitent un processus de fabrication complexe qui met en jeu un nombre important de machines. En réduisant les temps d'attente sur ces dernières, il est possible d'augmenter la productivité globale d'une usine. La résolution de ces problèmes d'ordonnancement devient vite très difficile dès lors que l'on ajoute quelques contraintes. Par exemple, des contraintes de précédence entre les tâches spécifient qu'une tâche ne peut pas commencer avant qu'une ou plusieurs autres tâches aient terminé.

Dans le monde de l'informatique, les problèmes d'ordonnancement sont également omniprésents. Encore une fois, les problèmes rencontrés sont de natures très variées. L'ordonnancement d'un code machine composé d'un ensemble d'instructions sur un processeur doit considérer les dépendances de donnée qui peuvent exister entre les instructions, ainsi que les différentes unités de calcul du processeur. En général, le but est de minimiser la durée d'exécution totale du code machine. Dans les systèmes temps-réel, il est souvent question de tâches périodiques contraintes par des dates d'échéance.

Ordonnancement temps-réel de tâches périodiques

Soient n tâches périodiques T_i, $i \in \{ 1,\ldots,n \}$, de périodes respectives P_i, c'est-à-dire si t_i est la date d'activation d'une des occurrence de T_i, alors la date d'activation de la prochaine occurrence de T_i est $t_i + P_i$. Supposons de plus que pour tout i, la durée d'exécution C_i de la tâche T_i est connue, et que toute occurrence de T_i doit terminer avant l'activation de l'occurrence suivante, c'est-à-dire dans un laps de temps de P_i à partir de son activation. Ainsi, si t_i est la date d'activation d'une occurrence de T_i, alors $D_i = t_i + P_i$ est sa date d'échéance. Il s'agit donc d'un problème temps-réel dur dans lequel les tâches ne doivent pas dépasser leurs échéances. Bien entendu, pour tout i nous avons $C_i \leq P_i$ sinon, le problème n'a pas de solution.

Dans l'analyse *Rate Monotonic* (RM) [33], on assigne des priorités fixes aux tâches. La priorité d'une tâche est d'autant plus forte que sa période d'activation est petite. On apprend également dans cette même analyse que la condition sur l'utilisation maximale du processeur :

$$\sum_{1 \leq i \leq n} \frac{C_i}{P_i} \leq n(2^{\frac{1}{n}} - 1), \tag{1.1}$$

est une condition nécessaire pour que le système soit ordonnançable, c'est-à-dire que les tâches respectent leurs échéances. La quantité $n(2^{\frac{1}{n}} - 1)$ qui borne l'utilisation du processeur dans (1.1) tend

en décroissant vers $ln\ 2 \approx 0.693$. Mais cette borne est loin d'être nécessaire. Il a en effet été montré dans [30] que l'utilisation du processeur peut monter jusqu'à 88% en moyenne, en se basant sur une analyse stochastique. Bien entendu, il est possible d'appliquer l'analyse RM lorsque les tâches sont pseudo-périodiques de pseudo période P_i. Dans ce cas, si t_i est la date d'activation d'une occurrence d'une tâche T_i, alors la date d'activation de la prochaine occurrence est supérieure ou égale à $t_i + P_i$. De même, si les durées d'exécution C_i ne sont pas les durées d'exécution exactes des tâches mais seulement des durées d'exécution pire-cas, l'analyse reste vraie. Cependant, la plupart des calculs de durée d'exécution pire-cas ne fournissent en général que des bornes fortement sur-estimées [48, 49, 15]. De plus, il est très peu probable, voire impossible, que toutes les durées d'exécution de toutes les tâches atteignent leur pire-cas en même temps. Dans ce cas, l'analyse RM, qui est statique, conduit à une forte sur-estimation de puissance de la plate-forme nécessaire à exécution de l'ensemble de tâches. Il s'agit du prix à payer pour garantir le respect des échéances.

Dans l'analyse *Earliest Deadline First* (EDF), ce ne sont plus des priorités fixes qui sont assignées aux tâches mais des priorités dynamiques : à tout instant, la priorité maximale est attribuée à la tâche dont la date d'échéance est la plus petite. Une telle politique nécessite donc l'usage d'un mécanisme de préemption du processeur, supposé instantané dans cette analyse. Dans ce cas, [33] montre que l'analyse EDF est optimale et que l'ensemble des tâches est ordonnançable si et seulement si :

$$\sum_{1 \leq i \leq n} \frac{C_i}{P_i} \leq 1. \tag{1.2}$$

Ainsi, la politique d'ordonnancement EDF permet une utilisation maximale du processeur lorsque les durées d'exécution sont connues. Cependant, elle appelle la même remarque que la politique RM : si les durées d'exécution ne sont pas connues, et que seules des durées d'exécution pire-cas sont fournies, l'utilisation du processeur pourra être très inférieure à celle qui est calculée de manière théorique à partir des durées d'exécution pire-cas, c'est-à-dire le membre gauche de (1.2).

1.2.2 Techniques d'ordonnancement avec contrôle d'un paramètre

Dans le paragraphe précédent, nous avons exposé quelques techniques d'ordonnancement temps-réel classiques. Ces techniques permettent de choisir un ordonnancement, c'est-à-dire un ordre d'exécution, adapté afin de satisfaire les contraintes initiales (échéances, précédence, etc...), ou encore optimiser certains paramètres (consommation mémoire, etc...). Ainsi, les seuls paramètres sur lesquels il était possible d'agir concernaient uniquement l'ordre d'exécution. Les techniques d'ordonnancement que nous allons voir ici agissent non seulement sur l'ordre d'exécution des tâches mais aussi sur d'autres paramètres tels que la fréquence du processeur ou encore le niveau de précision des calculs.

Les politiques d'ordonnancement de tâches périodiques RM et EDF (voir paragraphe précédent) ont connu différentes extensions en fonction de paramètres supplémentaires dans le système. Par exemple [41] propose, entre autres, un algorithme d'ordonnancement statique basé sur la politique RM et EDF dans le cadre de l'optimisation de la consommation électrique. L'approche repose sur le fait qu'en cas de sous charge du processeur, il est possible de réduire sa fréquence et sa tension, ce qui réduit sa consommation électrique. Ainsi, la fréquence du processeur est réduite tant que la borne

d'utilisation du processeur de l'analyse RM ou EDF (voir équations 1.1 et 1.2) n'est pas atteinte. Une telle approche permet de calculer la fréquence pour laquelle l'analyse RM ou EDF garantit le respect des échéances. Soit $f_1 < \ldots < f_m$ un ensemble ordonné de fréquences possibles pour le processeur cible. Reprenons le modèle de tâches périodiques présenté dans le paragraphe précédent. [41] fait l'hypothèse que les durées d'exécution sont proportionnelles à la fréquence du processeur, c'est-à-dire que la durée d'exécution de la tâche T_i à la fréquence f_j est donnée par $C_i \cdot f_j / f_m$. Dans ce cas, la fréquence f_j choisie est simplement donnée par :

$$\mathbf{max}\{\ \frac{C_i}{P_i}\ \leq\ \frac{f_j}{f_m} \cdot \alpha \mid 1 \leq j \leq m\}, \tag{1.3}$$

où α est la borne d'utilisation du processeur, c'est-à-dire $n(2^{\frac{1}{n}} - 1)$ pour l'analyse RM et 1 pour l'analyse EDF. Bien entendu, une telle approche n'atteindra pas des performances satisfaisantes lorsque les durées d'exécution ne peuvent pas être connues et que les valeurs C_i sont des durées d'exécution pire-cas. C'est pourquoi [41] propose aussi deux algorithmes de contrôle dynamique de la fréquence, dont il est question dans la suite de cet état de l'art.

1.2.3 Techniques d'ordonnancement dynamique sous incertitude

Nous l'avons vu, les techniques d'ordonnancement statique sont intéressantes lorsque les paramètres des tâches sont connus à l'avance. Dans le cas contraire, les techniques statiques ne sont pas performantes dans la mesure où elles ne sont pas capables de remettre en cause les choix initiaux afin de les adapter en fonction de ce qui se produit réellement dans le système. Les approches dynamiques ou adaptatives permettent de faire face à différentes formes d'incertitude : durées d'exécution ou périodes d'activation inconnues, voire même ensemble de tâches, et en particulier nombre de tâches, inconnu. Il est souvent possible de caractériser les valeurs incertaines d'un modèle par un encadrement — incertitude bornée — ou encore par un modèle stochastique [10]. Ces informations permettent à l'ordonnanceur de réaliser des prévisions sur l'avenir, augmentant ainsi la pertinence de ses décisions.

Slack Scheduling

Les techniques de *slack scheduling* étendent les politiques d'ordonnancement statique en se basant sur le constat suivant. Puisque les durées d'exécution pire-cas sont souvent sur-estimées par rapport aux durées d'exécution réelles, une partie du budget de temps pré-alloué statiquement est disponible dynamiquement au moment où la tâche termine. Le *slack scheduling* se fait donc en deux temps : tout d'abord, il convient d'évaluer le budget de temps libéré — *slack time* — par les tâches qui ont terminé avant leur échéance, et ensuite il faut utiliser ce budget de temps. [31] considère le même modèle de tâches que celui de Liu et Layland [33], augmenté d'un ensemble de tâches apériodiques considérées comme étant temps-réel molles. La politique d'ordonnancement RM de Liu et Layland est modifiée de façon à calculer le budget de temps qui peut être alloué aux tâches apériodiques sans remettre en cause le respect des échéances des tâches périodiques. [31] explique également comment prendre

en compte dynamiquement le budget de temps qui est pré-alloué statiquement par la politique d'ordonnancement RM, et qui est rendue disponible lorsque les tâches périodiques terminent avant leur échéance. L'algorithme proposé permet de minimiser le temps de réponse des tâches apériodiques. [20] propose différentes extensions comme le partage de ressources (autres que le processeur) entre les tâches ou encore des échéances arbitraires $D_i > P_i$: si la date d'activation d'une occurrence d'une tâche périodique T_i est t_i, alors son échéance est donnée par $t_i + D_i$. Dans [32], Thuel et Lehoczky considèrent un problème d'ordonnancement à priorités fixes d'un ensemble de tâches périodiques et apériodiques, toutes contraintes par des échéances strictes. L'idée est de n'accepter que les tâches apériodiques dont on a l'assurance qu'elles respecteront leur échéance, les autres étant rejetées par l'ordonnanceur. Le critère d'admission est basé sur un algorithme de *slack scheduling* qui permet de prendre en compte les durées d'exécution réelles des tâches.

Gain Time

Alors que le *slack scheduling* raffine uniquement la durée d'exécution des tâches qui ont déjà été exécutées, en remplaçant les prévisions — durées d'exécution pire-cas — par les durées d'exécution réelles, les techniques de *gain time* s'intéressent à la durée d'exécution du code qui reste à exécuter. Le but est de raffiner la durée d'exécution pire-cas d'une tâche en fonction de la connaissance de ses données d'entrée ainsi que de son état courant. A partir de la durée d'exécution pire-cas du code qui reste à exécuter et de la durée d'exécution réelle du code qui a déjà été exécuté, les choix d'ordonnancement peuvent être anticipés [24]. Pour que l'analyse de *gain time* soit possible, il est nécessaire de travailler à un niveau de granularité plus fin que celui de la tâche système, c'est-à-dire en s'immisçant directement dans le code applicatif de la tâche. Pour ce faire, la durée d'exécution pire-cas de la tâche est ré-évaluée au cours de son exécution, sur des points de programme particuliers appelés *gain points*. Le choix de ces *gain points* conditionne en grande partie l'efficacité du *gain time* dans la mesure où ralentir le système par une sur-instrumentation ne ferait que diminuer la qualité de service. L'objectif est donc de choisir des points du programme pertinents quant à la mise à jour de la durée d'exécution pire-cas.

```
if (a > 10)
  f_1(a); // 1000 unités de temps
else
  f_2(a); // 10   unités de temps
```

FIG. 1.2: Exemple de morceau de code.

Considérons par exemple le bloc de code de la figure 1.2. Si les durées d'exécution pire-cas des fonctions `f_1` et `f_2` sont très différentes, par exemple 1000 unités de temps pour `f_1` et seulement 10 pour `f_2`, il est intéressant d'insérer un *gain point* immédiatement après l'évaluation de la condition `a > 10`, afin d'informer l'ordonnanceur s'il reste au pire-cas 1000 unités de temps ou seulement 10 avant que le bloc de code termine. Plus généralement, il est intéressant d'insérer un *gain time* à chaque fois qu'une partie non négligeable d'incertitude sur la durée d'exécution pire-cas peut être levée [11]. Une autre optimisation concerne l'ordre d'exécution du code. En effet, ordonnancer en premier les

parties du code les plus incertaines, lorsque cela est possible, assure que les durées d'exécution pire-cas soient les plus précises possibles le plus tôt possible [22].

Contrôle dynamique de la fréquence d'horloge du processeur

Lorsque les durées d'exécution des tâches sont incertaines, la possibilité de contrôler d'autres paramètres que l'ordre d'exécution devient particulièrement intéressant pour donner une plus grande flexibilité au système en augmentant sa capacité d'adaptation. [41] propose par exemple d'adapter dynamiquement la fréquence d'horloge du processeur, sur la base d'un modèle de tâches périodiques et d'un ordonnancement de type RM ou EDF. Deux algorithmes sont proposés. Dans le premier appelé *Cycle-conserving*, la fréquence d'horloge initiale est issue d'un calcul statique, dans lequel les durées d'exécution sont supposées au pire-cas. Cette fréquence correspond donc à celle qui est calculée avec l'algorithme statique présenté également dans [41] (voir section 1.2.2). Durant l'exécution, si l'occurrence d'une tâche T_i termine en avance par rapport à sa durée d'exécution pire-cas C_i, la valeur C_i est remplacée par la durée d'exécution réelle C_i^* dans le calcul de la fréquence basé sur l'analyse EDF, c'est-à-dire dans les équation 1.3. Ainsi, le budget du processeur qui a été économisé par la terminaison de T_i avant sa durée d'exécution pire-cas est intégré dynamiquement dans le calcul de la fréquence courante. Cependant, il se peut que la durée d'exécution de la prochaine occurrence de T_i soit égale au pire-cas C_i. Ainsi, à l'instant de la prochaine activation de T_i, la valeur C_i est restituée dans l'équation 1.3, à la place de C_i^*. Le principe est donc très proche de du *slack scheduling*, à la différence près que le budget de temps économisé par les tâches qui terminent avant leur durée d'exécution pire-cas est utilisé pour baisser la fréquence du processeur et non ordonnancer des tâches non critiques.

Dans le second algorithme de contrôle dynamique de fréquence appelé *Look-ahead*, [41] va encore plus loin en proposant une heuristique très intéressante. Cette dernière part du principe que les tâches terminent presque toujours bien avant leur durée d'exécution pire-cas. La fréquence initiale est calculée de telle sorte qu'il soit toujours possible de respecter les échéances, en l'augmentant si besoin est par la suite. Cette idée est également à la base du calcul dynamique de la fréquence d'horloge. A la différence de l'algorithme *Cycle-conserving*, la fréquence initiale ne peut pas être maintenue si les durées d'exécutions réelles sont toutes égales aux durées d'exécution pire-cas. En effet, dans ce cas, l'ordonnanceur sera forcé d'augmenter la fréquence d'horloge dans la mesure où il démarre avec une fréquence volontairement basse. L'ordonnanceur *Look-ahead* anticipe ainsi le fait que les durées d'exécution réelles sont presque toujours inférieures aux durées d'exécution pire-cas. Le système reste néanmoins sûr, c'est-à-dire que les échéances sont toujours respectées, puisque l'ordonnanceur a toujours la possibilité d'augmenter la fréquence si les durées d'exécution réelles sont trop élevées. En pratique, l'ordonnanceur *Look-ahead* se montre plus performant que l'ordonnanceur *Cycle-conserving*, par une réduction plus agressive de la fréquence d'horloge.

Imprecise computation

Les approches dites d'*imprecise compuation* [36, 34, 35] étendent les techniques d'ordonnancement temps-réel dur de type RMA ou EDF [33]. Même dans un système de type temps-réel dur, il

est possible que certains calculs effectués par certaines tâches puissent être dégradés sans remettre en cause le bon fonctionnement du système. Pour ce faire, la durée d'exécution de chaque tâche périodique T_i est décomposée en deux parties : la première donne la durée d'exécution minimale C_i^m de T_i pour obtenir un résultat acceptable. Le système est considéré comme ordonnançable si les tâches respectent leur échéance, et si la durée d'exécution de chaque tâche T_i est supérieure ou égale à C_i^m. Lorsque la durée d'exécution d'une instance d'une tâche périodique est inférieure à sa durée d'exécution maximale $C_i \geq C_i^m$, le résultat produit par T_i est considéré comme imprécis. Cette imprécision est mesurée par ε, qui est une fonction décroissante de la durée d'exécution de T_i, nulle en C_i. Deux types de tâches sont proposées dans [17] : pour les tâches de type N, seule l'erreur moyenne est pertinente ; pour les tâches de type C, l'erreur produite par les différentes instance d'une tâche est accumulée.

Temps-réel souple

Lorsqu'il existe une part d'incertitude sur les durées d'exécution dans le système, les analyses de type pire-cas des approches temps-réel critique conduisent à de mauvaises performances quant à l'utilisation du processeur. Pour cette raison, de nombreux travaux de type temps-réel souple ont été menés pour traiter les problèmes d'ordonnancement sous incertitude.

Dans [37], Stankovic et al. partent de l'hypothèse que les tâches du système sont paramétrées par un niveau de qualité entier (*QoS level*). Certaines applications — multimédia [13, 38], 3D [29], WEB, télécommunication — supportent une réduction progressive de la précision de leurs calculs (*graceful degradation*). Cette dernière s'accompagne d'une réduction de la consommation en ressources de calcul et/ou de mémoire. Il est important de noter que la fonctionnalité de l'application n'est pas affectée par cette dégradation, seule la qualité de service diminue. Les niveaux de qualité du modèle proposé dans [37] correspondent donc à différentes qualités de service possibles. Ils peuvent aussi contrôler l'admission d'une tâche : dans ce cas le niveau de qualité minimal correspond à la non-admission de la tâche. L'approche de Stankovic et al. constitue un cadre très général de conception des systèmes temps-réel souples, basée sur un ordonnanceur et un gestionnaire de qualité qui visent à optimiser les performances globales du système. Cette optimisation est faite par le biais de l'observation de paramètres globaux comme le nombre d'échéances ratées. Le gestionnaire de qualité de service proposé détermine le niveau de qualité des tâches à l'aide de techniques de contrôles classique comme les contrôleurs PID [21].

Dans [50], Wüst et al. s'intéressent au contrôle de la qualité de service d'un décodeur vidéo. Ils partent du principe que les applications multimédia comme le décodage vidéo ont des durées d'exécution très variables, fortement dépendantes des données d'entrée. Pour restituer correctement une séquence vidéo, les images qui la constituent (ou *frames*) doivent être affichées à des dates précises. Par exemple, pour une vidéo encodée à 25 images par seconde, une image doit être produite toutes les 40 ms. Dans les situations de surcharge du processeur, le décodeur ne peut plus satisfaire les contraintes temps-réel. La pratique courante est d'utiliser un tampon mémoire en entrée du processus de décodage pour absorber les surcharges temporaires du processeur. Lorsque le tampon est plein, certaines images ne sont pas décodées (*frame skip* [28]) afin de continuer à satisfaire les contraintes de temps, au détriment de qualité de la vidéo. [50] propose de réduire progressivement la qualité des

algorithmes de décodage afin d'éviter que le taux de *frame skips* ne devienne trop élevé lorsque le processeur est surchargé. Le décodeur comporte quatre niveaux de qualité, et [50] présente plusieurs algorithmes de contrôle de ces niveaux de qualité basés sur une modélisation en terme de chaînes de Markov et des techniques d'apprentissage. Dans [40], Steffens et al. montrent comment intégrer cette technique au sein d'un système complet composé de plusieurs tâches.

1.3 Notre approche

Nous considérons que les approches temps-réel souple ne sont pas toujours suffisantes, même dans le contexte des applications multimédia [26, 14]. En effet, pour certaines applications, rater une échéance peut conduire à une dégradation drastique de la qualité de service. C'est le cas par exemple des applications de décodage vidéo. En effet, lorsqu'une image n'est pas décodée, non seulement la fluidité d'image est affectée, mais la qualité du reste de la séquence est également dégradée du fait de l'utilisation d'un encodage différentiel. Nous proposons ici une approche innovante dans laquelle un contrôleur est embarqué directement dans l'application. Celui-ci assure conjointement le respect des propriétés de type temps-réel critique (respect des échéances), et le respect des propriétés de type temps-réel souple (optimisation de l'utilisation du processeur).

Nous considérons que l'application effectue des traitements cycliques sur un flux de données d'entrée. L'enchaînement des traitements, appelés actions, est décrit par un graphe de précédence dont il est possible d'extraire tous les ordonnancements possibles. Les actions sont implémentées par des fonctions écrites en langage C, et sont paramétrées par un niveau de qualité qui conditionne le degré de précision des calculs ainsi que les durées d'exécutions. Nous considérons une plate-forme monoprocesseur sur laquelle l'application est seule à s'exécuter (plate-forme nue). Nous supposons de plus qu'il est possible d'extraire les durées d'exécution moyennes et pire-cas des actions, pour tous les niveaux de qualité, grâce à des analyses statique et/ou par l'observation de traces d'exécution réelles (techniques de *profiling*). Cette plate-forme doit aussi fournir une horloge temps-réel précise et accessible sans sur-coût, comme par exemple un compteur de cycle comme il en existe sur nombreux processeurs dans le domaine de l'embarqué temps-réel. Les contraintes temps-réel sont données par des échéances sur les actions. Elles donnent une borne supérieure de la différence entre date de démarrage d'un cycle de calcul et la la date de terminaison des actions, c'est-à-dire la date de production des sorties de l'application. Un outil permet de générer automatiquement l'application contrôlée à partir du graphe de précédence G, des durées d'exécutions moyennes C^{av} et pire-cas C^{wc} et des échéances D (voir figure 1.3). Le contrôleur est paramétré par une politique de contrôle dont le but est d'assurer les exigences de qualité de service. Nous considérons trois types de propriétés.

Sûreté. Il s'agit du respect des contraintes temps-réel, c'est-à-dire des échéances.

Optimalité. Le critère d'optimalité sera l'utilisation du budget de temps alloué à l'application par les échéances. En maximisant cette utilisation, nous permettons à l'application d'activer les niveaux de qualité les plus élevés possibles.

Régularité. Nous considérons que la qualité de service finale dépend aussi de la régularité des niveaux de qualité choisis. Il s'agit d'une propriété essentielle pour les applications multimédia,

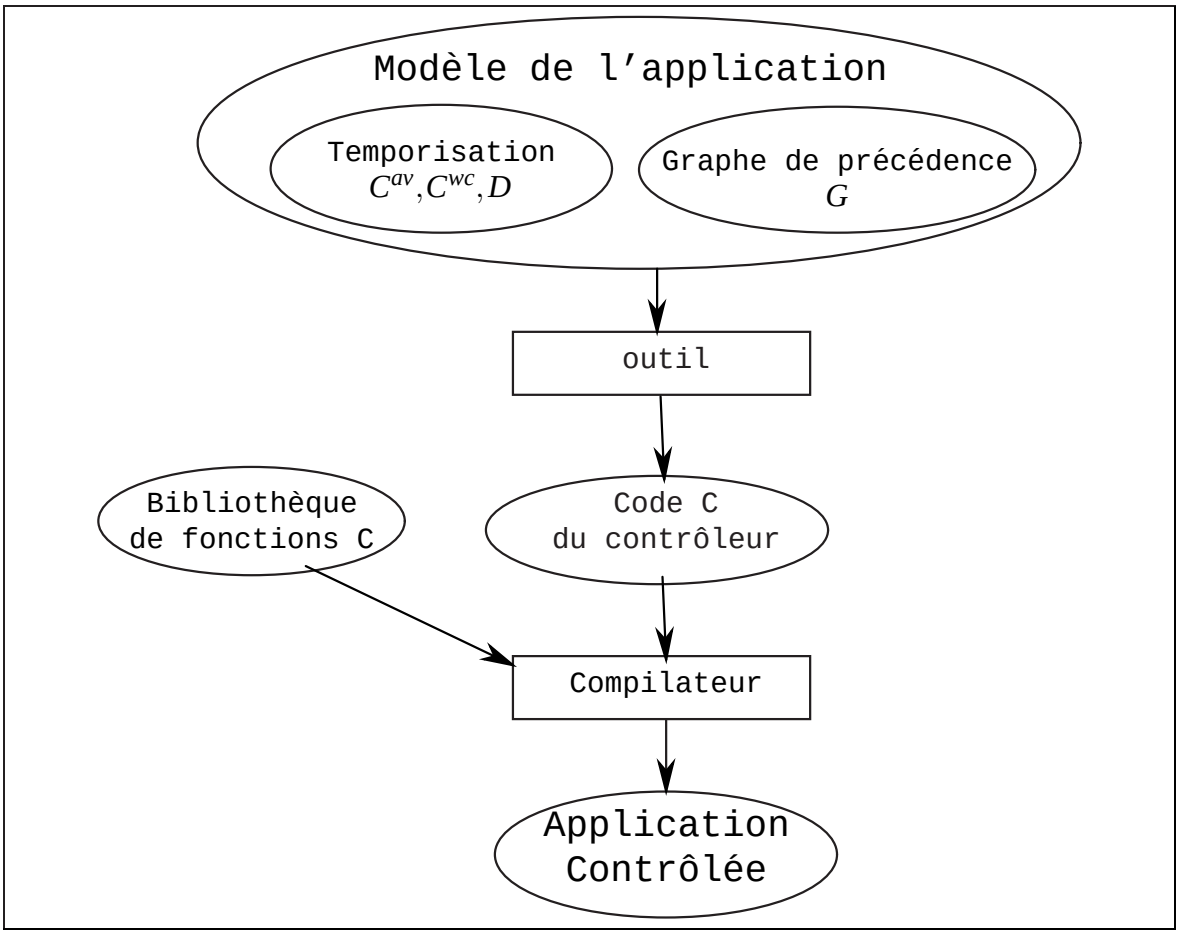


FIG. 1.3: Génération de l'application contrôlée.

et plus particulièrement les applications vidéo. En effet, il a été montré que la qualité finale d'une séquence vidéo, c'est-à-dire celle qui est perçue par l'utilisateur, dépend non seulement de la qualité individuelle de chaque image (ou *frame*) qui compose cette séquence, mais aussi et surtout de sa fluctuation et de son niveau minimal [45, 47].

Dans ce document, nous allons voir qu'en agissant directement à grain fin dans l'application, nous sommes capables de contrôler efficacement l'ordre d'exécution des actions et leur niveau de qualité, de sorte que leur dates de terminaison soient très proches de leur échéance. Ainsi, nous assurons non seulement le respect des échéances, mais aussi une utilisation presque totale de la puissance de la plate-forme d'exécution. Cette propriété, combinée à la régularité des niveaux de qualité, garantit une maximisation de la qualité de service. Ce travail contribue donc à réduire le fossé qui sépare le monde du temps-réel critique de celui du temps-réel souple. Si certains efforts ont déjà été menés dans ce sens, au sein travaux effectués essentiellement dans les années 90 et 2000, aucun n'était jusque là parvenu à réunir autant de caractéristiques issues des deux approches : respect des échéances, utilisation élevée du processeur, régularité des niveaux de qualité grâce à l'utilisation des durées d'exécution moyennes. Nous pensons que notre approche peut servir de base à une méthodologie globale développement de composants à la fois flexibles (adaptatifs) et robustes, nécessaires pour la conception des systèmes embarqués de demain.

Plan de ce document

Ce manuscrit est organisé de la façon suivante. Tout d'abord, nous feront une présentation incrémentale du problème de contrôle auquel nous nous intéressons ici dans le chapitre 2. Ensuite, nous proposerons une solution basée sur un algorithme de contrôle générique. Nous donnerons plusieurs instanciations de ce contrôleur, correspondant à différentes politiques de contrôle dont nous donnerons les propriétés dans le chapitre 3. Le chapitre 4 sera consacré à l'étude des algorithmes qui permettent d'implémenter efficacement le contrôleur et les politiques de contrôles proposées. Le chapitre 5 concernera les résultats expérimentaux obtenus dans le cas d'une application d'encodage vidéo. Nous montrerons que les expériences sont en accord avec les résultats théoriques du chapitre 3, et qu'ils confirment l'intérêt de la méthode. Pour finir, la conclusion fera un bilan de ce travail et en présentera les perspectives.

Notations et définitions de base

Opérateurs logiques

On appele *prédicat* sur un ensemble A tout fonction f de la forme $f : A \rightarrow \{ \ vrai, faux \ \}$. Dans la suite, nous utiliserons les opérations suivantes sur les prédicats :

$p_1 \wedge p_2$	conjonction des prédicats p_1 et p_2 (“et”)
$p_1 \vee p_2$	disjonction des prédicats p_1 et p_2 (“ou”)
$p_1 \Rightarrow p_2$	implication du prédicat p_2 par le prédicat p_2
$p_1 \Leftrightarrow p_2$	équivalance entre les prédicats p_1 et p_2
$\neq p$	négation du prédicat p.

Ensembles

$\mathbb{N}$	ensemble des entiers positifs ou nuls
$\mathbb{Z}$	ensemble des entiers relatifs
$\mathbb{R}^+$	ensemble des réels positifs ou nuls
$\mathbb{R}$	ensemble des réels
$\|A\|$	cardinal de l’ensemble A, c’est-à-dire : $+\infty$ si A n’est pas fini 0 si $A = \emptyset$ n si A est un ensemble fini de la forme $A = \{ \ a_1, \ldots, a_n \ \}$
$A_1 \times A_2$	produit cartésien de l’ensemble A_1 par l’ensemble A_2, c’est-à-dire : $A_1 \times A_2 = \{ \ (a_1, a_2) \mid a_1 \in A_1 \ \wedge a_2 \in A_2 \ \}$
A^n	produit cartésien $\underbrace{A \times A \times \ldots \times A}_{n \text{ fois}}$
$A \setminus B$	soustraction ensembliste : $A \setminus B = \{ \ a \in A \mid a \notin B \ \}$.

Séquences

Soit A un ensemble fini. Une séquence finie α d'éléments de A est un n-uplet $\alpha \in A^n$ d'éléments de A, où n désigne la *longueur* de α, notée $|\alpha|$. Nous notons ε la séquence de longueur nulle. Une sequence α de longueur non nulle sera notée $\alpha = \alpha(1)\alpha(2)\ldots\alpha(|\alpha|)$. Elle vérifie pour tout i $\alpha(i) \in A$. Nous définissons alors les notations suivantes, où α désigne une séquence de longueur non nulle :

A^*	désigne l'ensemble des séquences finies d'élément de A, c'est-à-dire : $A = \{\ \varepsilon\ \} \cup \bigcup_{n>0} A^n$
$\{\alpha\}^n$	la séquence de longueur $n\|\alpha\|$ définie par $\{\alpha\}^n = \underbrace{\alpha\alpha\ldots\alpha}_{n \text{ fois}}$
$\alpha[i,j]$	la sous-séquence $\alpha[i,j] = \alpha(i)\ldots\alpha(j)$ de longueur $j-i+1$
${}^i\alpha$	le préfixe de longueur i de α, c'est-à-dire ${}^i\alpha = \alpha[1,i]$ (${}^i\varepsilon = \varepsilon$)
α^i	le suffixe de longueur $\|\alpha\| - i + 1$, c'est-à-dire $\alpha^i = \alpha[i,\|\alpha\|]$ ($\varepsilon^i = \varepsilon$)
$\mathsf{ens}(\alpha)$	l'ensemble des éléments de α, c'est-à-dire $\mathsf{ens}(\alpha) = \{\ \alpha(1),\ldots,\alpha(\|\alpha\|)\ \}$ par convention, nous définissons $\mathsf{ens}(\varepsilon) = \emptyset$.

Fonctions partielles

Soit $f : A \to B$ une fonction quelconque. Nous dirons que f est une fonction ***partielle*** si elle n'est pas définie sur A tout entier, et ***totale*** dans le cas contraire. Nous introduisons les notations suivantes, dans lesquelles f est une fonction du type $f : A \to B$, $a, a_1, a_2, \ldots, a_n$ sont des éléments de A, $b, b_1, b_2, \ldots, b_n$ sont des éléments de B, et α une séquence d'éléments de A :

$f(a) = \perp$	signifie que f n'est pas définie en $a \in A$
$\mathsf{dom}(f)$	domaine de définition de f, c'est-à-dire $\mathsf{dom}(f) = \{\ a \mid f(a) \neq \perp\ \}$
$f[a_1 \to b_1, \ldots, a_n \to b_n]$	extension de la fonction f en une fonction dans laquelle l'image de $a_i \in A$ est $b_i \in B$, c'est-à-dire : $f[a_1 \to b_1, \ldots, a_n \to b_n](a') = \begin{cases} b_i \text{ si } a' = a_i \\ f(a') \text{ sinon} \end{cases}$
$[a_1 \to b_1, \ldots, a_n \to b_n] =$ $f[\alpha \to b]$	$\perp[a_1 \to b_1, \ldots, a_n \to b_n]$ extension de la fonction f en une fonction dans laquelle les éléments de la séquence α, c'est-à-dire : $f[\alpha \to b] = f[\alpha(1) \to b, \ldots, \alpha(\|\alpha\|) \to b]$.

Relations binaires

Soit A un ensemble quelconque. Nous appelons ***relation binaire*** sur A tout sous-ensemble $\mathfrak{R}$ de $A \times A$. Soit $\mathfrak{R}$ une relation binaire sur l'ensemble A. Nous noterons $a\mathfrak{R}b$ pour $(a,b) \in \mathfrak{R}$. La relation $\mathfrak{R}$ est dite :

réflexive	si pour tout $a \in A$, nous avons $a \Re a$
irréflexive	si pour tout $a \in A$, nous n'avons pas $a \Re a$
symétrique	si pour tout $(a,b) \in A^2$, nous avons $a \Re b \Rightarrow b \Re a$
antisymétrique	si pour tout $(a,b) \in A^2$, nous avons $(a \Re b \wedge b \Re a) \Rightarrow a = b$
transitive	si pour tout $(a,b,c) \in A^3$, nous avons $(a \Re b \wedge b \Re c) \Rightarrow a \Re c$.

Nous utiliserons la terminologie suivante :

relation d'équivalance	relation réflexive, symétrique, et transitive
ordre partiel strict	relation irréflexive, antisymétrique et transitive.

Graphes de précédence

Un graphe de précédence est un couple $G = (A, \prec)$ où A est un ensemble fini et $\prec$ un ordre partiel strict sur A. Pour plus de détails sur les graphes de précédence, se référer à la section 2.1.1. Nous donnons ici les notations relatives aux graphes de précédence :

G/B	désigne le graphe restreint au sous-ensemble $B \subseteq A$, c'est-à-dire : $G/B = (B, \prec \cap (B \times B))$
$G_1 G_2$	désigne la concaténation de $G_1 = (A_1, \prec_1)$ avec $G_2 = (A_2, \prec_2)$, c'est-à-dire $G_1 G_2 = (A_1 \cup A_2, \prec_1 \cup \prec_2 \cup A_1 \times A_2)$
$\mathsf{pred}_G(a)$	désigne les prédécesseurs immédiats de a, c'est-à-dire : $\mathsf{pred}_G(a) = \{\ b \in A \mid (b \prec a) \wedge (\nexists c \in A\ .\ b \prec c \wedge c \prec a)\ \}$
$\mathsf{succ}_G(a)$	désigne les successeurs immédiats de a, c'est-à-dire : $\mathsf{succ}_G(a) = \{\ b \in A \mid (a \prec b) \wedge (\nexists c \in A\ .\ a \prec c \wedge c \prec b)\ \}$.

CHAPITRE 2

Contrôle de qualité

Ce premier chapitre concerne la définition du problème de contrôle que nous cherchons à résoudre dans ce travail de thèse. Une solution et une implémentation efficace seront proposées dans les chapitres suivants. La présentation du problème est faite de manière incrémentale. Nous considérons une machine monoprocesseur tout au long de ce document.

Nous exposons tout d'abord un problème simple et connu. Il s'agit de l'ordonnancement d'un ensemble de tâches, ou *actions* dans notre terminologie, dont les durées d'exécution sont connues, et dont l'exécution est contrainte par un graphe de précédence et des dates d'échéances absolues.

Nous considérons une première extension du problème, dans laquelle les actions sont paramétrées par un *niveau de qualité*, qui est un paramètre entier ayant un ensemble fini de valeurs possibles. Dans ce problème d'ordonnancement, les durées d'exécution sont toujours connues, mais dépendent du paramètre de qualité : les durées d'exécution sont des fonctions croissantes du niveau de qualité. A partir de ce modèle, le problème d'ordonnancement est de déterminer un ordre d'exécution et une affectation de qualité tels que les actions respectent les contraintes de précédence et les échéances, et tels que la qualité globale soit maximale, dans un sens qui sera précisé.

Le problème précédent est une nouvelle fois étendu. Dans cette dernière extension, les durées d'exécution sont inconnues, et sont appelées durées d'exécution *réelles*. Ces dernières sont cependant bornées par des durées d'exécution pire-cas qui, elle, sont connues. Nous considérons également qu'il est possible de connaître la durée d'exécution réelle d'une action une fois que son exécution est terminée. Ainsi, le dernier problème d'ordonnancement est un problème de contrôle dynamique.

2.1 Ordonnancement pour des durées d'exécution connues

Nous présentons ici un problème d'ordonnancement sur une machine monoprocesseur. Il s'agit d'ordonnancer un ensemble fini A de tâches (ou *actions*), dont les durées d'exécution sont connues et données par la fonction $C : A \to \mathbb{R}^+$. L'ordre d'exécution des actions est contraint par un graphe de précédence G, et des dates d'échéance absolues données par la fonction $D : A \to \mathbb{R}^+ \cup \{ +\infty \}$. Résoudre le problème revient à trouver un ordre d'exécution qui respecte les contraintes de précédence et tel que pour toute action a sa date de terminaison soit inférieure à son échéance $D(a)$.

2.1.1 Notion de graphe de précédence

Un graphe de précédence est utilisé pour modéliser les dépendances de donnée qui peuvent exister dans l'application. Le graphe de précédence est un ordre partiel sur l'ensemble d'actions.

DÉFINITION 2.1 *Soit A un ensemble fini. Un* **graphe de précédence** *G est un couple $G = (A, \prec)$, où $\prec \subseteq A \times A$ est un ordre partiel strict sur A. Nous dirons que $a \in A$* **dépend** *de $b \in A$ dans G si $b \prec a$. Nous dirons que a et b sont* **indépendantes** *si a ne dépend pas de b et b ne dépend pas de a.*

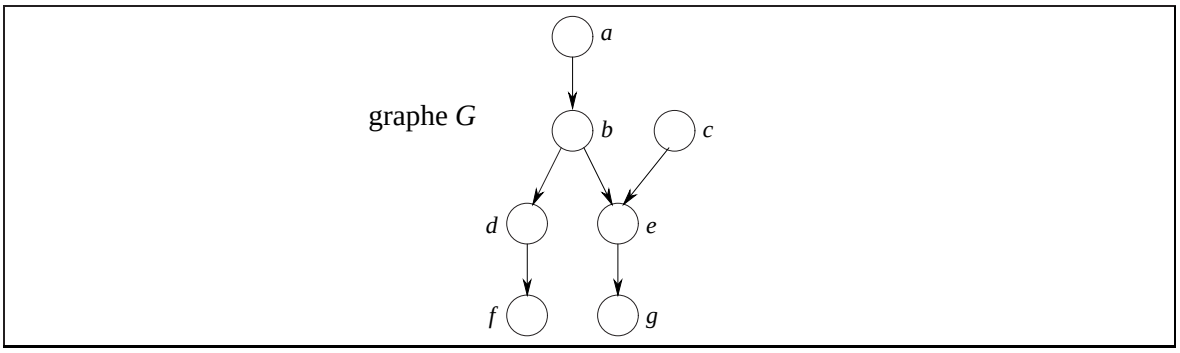


FIG. 2.1: Exemple de graphe de précédence.

DÉFINITION 2.2 *Nous introduisons à présent quelques définitions et notations utiles concernant les graphes de précédence :*

G/B	*désigne le graphe* **restreint**, *ou* **restriction** *de G, au sous-ensemble $B \subseteq A$, c'est-à-dire $G/B = (B, \prec \cap (B \times B))$*
G_1G_2	*désigne la* **concaténation** *de $G_1 = (A_1, \prec_1)$ avec $G_2 = (A_2, \prec_2)$, c'est-à-dire :* $G_1G_2 = (A_1 \cup A_2, \prec_1 \cup \prec_2 \cup A_1 \times A_2)$
$\mathsf{pred}_G(a)$	*désigne les* **prédécesseurs immédiats** *de a dans G, c'est-à-dire :* $\mathsf{pred}_G(a) = \{ b \in A \mid (b \prec a) \wedge (\nexists c \in A \,.\, b \prec c \wedge c \prec a) \}$
$\mathsf{succ}_G(a)$	*désigne les* **successeurs immédiats** *de a dans G, c'est-à-dire :* $\mathsf{succ}_G(a) = \{ b \in A \mid (a \prec b) \wedge (\nexists c \in A \,.\, a \prec c \wedge c \prec b) \}$.

Par soucis de concision dans la représentation graphique d'un graphe de précédence $G = (A, \prec)$, nous ne tracerons que les arcs $a \rightsquigarrow b$, où désigne $\rightsquigarrow$ la relation *prédécesseur-successeur immédiat*, définie par :

$$a \rightsquigarrow b \;\Leftrightarrow\; a \in \mathsf{pred}_G(b).$$

La relation $\rightsquigarrow$ ne conserve que les arcs de précédence immédiats entre les actions. Il est possible de reconstruire l'ordre partiel $\prec$ à partir de $\rightsquigarrow$ en calculant la fermeture transitive de $\rightsquigarrow$:

$$a \prec b \;\Leftrightarrow\; \exists (a_1, \ldots, a_n)\,.\,(a_1 = a \;\wedge\; a_n = b \;\wedge\; \forall i \in \{\, 1, \ldots, n-1 \,\}\,.\, a_i \rightsquigarrow a_{i+1}).$$

EXEMPLE 2.1 *Un exemple de graphe de précédence est donné par le graphe G de la figure 2.1. Ce graphe $G = (A, \prec)$ est tel que $A = \{\, a, b, c, d, e, f, g \}$. La représentation graphique décrit la relation prédécesseur-successeur immédiats $\rightsquigarrow$ associée à $\prec$. Celle-ci est donnée par :*

$$\rightsquigarrow = \{\, (a,b), (b,e), (b,d), (c,e), (d,f), (e,g) \,\}.$$

Ainsi, la relation $\prec$, qui est la fermeture transitive de $\rightsquigarrow$, est donnée par :

$$\begin{aligned} \prec = \{ \quad & (a,b), (a,d), (a,e), (a,f), (a,g), \\ & (b,e), (b,d), (b,f), (b,g), \\ & (c,e), (c,g), \\ & (d,f), (e,g) \,\}. \end{aligned}$$

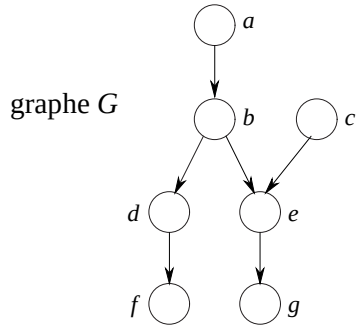
2.1.2 Définition du problème

Nous allons définir le problème de l'ordonnancement avec attributs constants. Nous commençons par quelques définitions à propos du modèle — appelé *système simple* — considéré pour ce problème. Un *système simple* est un ensemble fini d'actions contraintes par un graphe de précédence, et dont les durées d'exécution et les échéances sont des constantes connues.

DÉFINITION 2.3 *Nous appelons* **système simple** *un triplet (G, C, D) tel que :*

1. *$G = (A, \prec)$ est un graphe de précédence sur l'ensemble d'*actions *A.*
2. *$C : A \to \mathbb{R}^+$ est une fonction de* durée d'exécution.
3. *$D : A \to \mathbb{R}^+ \cup \{\, +\infty \,\}$ est une fonction d'*échéance.

DÉFINITION 2.4 *Soit A un ensemble fini d'actions. Nous appelons* **séquence d'actions** *toute séquence finie de la forme $\alpha = \alpha(1)\alpha(2)\ldots\alpha(n)$ telle que pour tout $i \in \{\, 1, \ldots, n \,\}$, $\alpha(i)$ est un élément de A. Nous appelons n la* **longueur** *de α. Nous notons ε la séquence d'actions vide, de longueur nulle par définition. Pour finir, nous notons $\mathsf{ens}(\alpha)$ l'ensemble des actions qui composent la séquence α, c'est-à-dire $\mathsf{ens}(\alpha(1)\ldots\alpha(n)) = \{\, \alpha(1), \ldots, \alpha(n) \,\}$ et $\mathsf{ens}(\varepsilon) = \emptyset$.*



action x	$C(x)$	$D(x)$
a	1	1
b	2	$+\infty$
c	1	$+\infty$
d	2	9
e	1	$+\infty$
f	3	$+\infty$
g	2	8

FIG. 2.2: Exemple de système simple.

Une séquence d'actions fixe un ordre d'exécution particulier pour une partie des actions du modèle. Dans la définition qui suit, nous introduisons les opérations nécessaires à la manipulation des séquences.

DÉFINITION 2.5 *Soit A un ensemble fini d'actions et $\alpha = \alpha(1)\dots\alpha(n)$ une séquence d'actions de longueur n. Soit i et j deux indices tels que $1 \leq i < j \leq n$. Dans la suite, la* **sous-séquence** *de α notée $\alpha[i,j]$ est définie par $\alpha[i,j] = \alpha(i)\alpha(i+1)\dots\alpha(j)$. Par convention, nous avons $\varepsilon[i,j] = \varepsilon$ pour tout indices i et j.*

Nous appelons **préfixe** *de α de longueur i la séquence d'actions $\alpha[1,i]$, notée ${}^{i}\alpha$. Symétriquement, nous appelons* **suffixe** *de α de longueur $n-i+1$ la séquence d'actions $\alpha[i,n]$, notée α^{i}.*

DÉFINITION 2.6 *Soit $G = (A, \prec)$ un graphe de précédence. Nous appelons* **trace** *de G toute séquence α d'éléments de A telle que :*

1. *ses éléments sont distincts, c'est-à-dire $i \neq j \Rightarrow \alpha(i) \neq \alpha(j)$;*
2. *pour tout $i \in \{1,\dots,|\alpha|\}$, $\mathsf{ens}({}^{i}\alpha)$ est fermé en arrière, c'est-à-dire que si $a \in \mathsf{ens}({}^{i}\alpha)$ et $a' \prec a$, alors $a' \in \mathsf{ens}({}^{i}\alpha)$.*

Si de plus la trace α est de longueur $|A|$, alors elle est appelée **ordonnancement** *de G.*

Soit α une trace de G et α' une séquence d'éléments de A. Nous dirons que α' est un **prolongement** *de α dans le graphe G si $\alpha\alpha'$ est un ordonnancement de G. Nous appellerons graphe* **résiduel** *du graphe G après l'exécution de la trace α, le graphe $G/(A \setminus \mathsf{ens}(\alpha))$.*

Une trace est un ordre d'exécution d'une partie des actions d'un graphe de précédence G, qui respecte les contraintes de précédence. Un prolongement d'une trace α est une séquence d'actions α' qui complète α de telle sorte que la concaténation $\alpha\alpha'$ est un ordonnancement. Le graphe résiduel du graphe d'origine G après l'exécution de la trace α correspond à la restriction du graphe G aux actions qui restent à exécuter, c'est-à-dire les actions $A \setminus \mathsf{ens}(\alpha)$. Autrement dit, un prolongement α' de la trace α dans G est un ordonnancement du graphe résiduel après avoir exécuté α.

EXEMPLE 2.2 *Considérons l'ensemble d'actions $A = \{ a,b,c,d,e,f,g \}$, et le graphe de précédence G de la figure 2.2. La séquence $\alpha_1 = adf$ n'est pas une trace de G, car l'ensemble $\{a,d\}$ n'est pas*

fermé en arrière ($b \prec d$). La séquence $\alpha_2 = abdf$ est une trace G, mais n'est pas un ordonnancement, car $|abdf| = 4 < |A| = 7$. Un ordonnancement est donné, par exemple, par la séquence $\alpha_3 = abdfceg$. Ainsi, la séquence $\alpha_4 = ceg$ prolonge la trace $\alpha_2 = abdf$ dans le graphe G.

NOTATIONS 2.1 *Afin de simplifier l'exposé qui va suivre, nous introduisons les notations suivantes. Soit (G,C,D) un système simple et α une séquence d'actions quelconque. Dans la suite, nous désignerons par $C(\alpha)$ la durée d'exécution totale des actions de α, c'est-à-dire :*

$$C(\alpha) = \sum_{1 \leq i \leq |\alpha|} C(\alpha(i)).$$

Nous désignerons par $D(\alpha)$ l'échéance de la dernière action de α, c'est-à-dire :

$$D(\alpha) = D(\alpha(|\alpha|)).$$

DÉFINITION 2.7 *Soit (G,C,D) un système simple et α un ordonnancement de G. Nous dirons qu'un ordonnancement α est* **réalisable** *si :*

$$\forall k \in \{\, 1, \ldots, |\alpha| \,\} \;.\; D(^k\alpha) \geq C(^k\alpha).$$

Un ordonnancement α d'un système simple $(G = (A, \prec), C, D)$ est réalisable si toutes les actions de A respectent leur échéance. Nous dirons qu'une action $a \in A$ respecte son échéance $D(a)$ dans l'ordonnancement α si la date de terminaison de a est inférieure ou égale à son échéance $D(a)$. Le calcul de la date de terminaison de a dans l'ordonnancement α est basé sur une exécution des actions telle que :

- l'exécution commence à la date 0 ;
- les actions sont exécutées dans l'ordre imposé par α ;
- il n'y a pas de temps mort entre l'exécution de deux actions consécutives.

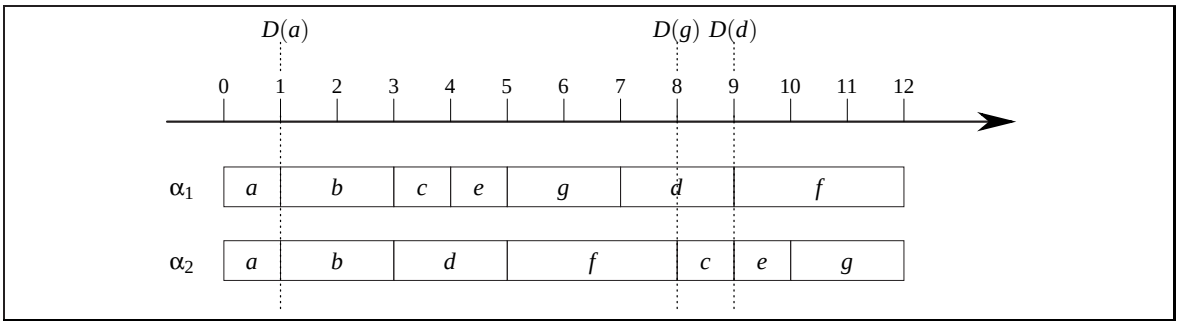


FIG. 2.3: Diagramme de Gantt des ordonnancements α_1 et α_2.

EXEMPLE 2.3 *Considérons à nouveau l'exemple donné par la figure 2.2. Soit $\alpha_1 = abcegdf$ et $\alpha_2 = abdfceg$ deux séquences d'actions. Il est facile de vérifier que α_1 et α_2 sont des ordonnancements du graphe G donné par la figure 2.2. La figure 2.3 donne le diagramme de Gantt des deux*

ordonnancements α_1 et α_2. Nous remarquons que l'ordonnancement α_2 n'est pas réalisable. En effet, nous avons :

$$D(abdfceg) = D(g) = 8 \ < \ 12 = C(a)+C(b)+C(d)+C(f)+C(c)+C(e)+C(g) = C(abdfceg).$$

Au contraire, l'ordonnancement α_1 est réalisable. Vérifions que toutes les échéances de α_1 sont respectées. Tout d'abord, remarquons que les échéances valant $+\infty$ sont forcément respectées. Nous vérifions donc uniquement les échéances inférieures à $+\infty$, c'est-à-dire $D(a)$, $D(g)$ et $D(d)$:

$$\begin{aligned} D(a) = 1 &\geq 1 = C(a) \\ D(abceg) = D(g) = 8 &\geq 7 = C(abceg) \\ D(abcegd) = D(d) = 9 &\geq 9 = C(abcegd). \end{aligned}$$

PROBLÈME 2.1 (ordonnancement d'un système simple) *Soit (G,C,D) un système simple. Le problème est de trouver un ordonnancement α tel que α soit un ordonnancement réalisable de (G,C,D).*

Dans la suite, nous allons voir comment formuler le problème sous forme d'un ensemble de contraintes linéaires sur des variables entières et réelles. Nous n'utiliserons pas ce formalisme pour résoudre le problème, mais il constitue un cadre général pour exprimer et résoudre des problèmes d'optimisation combinatoire, et plus particulièrement des problèmes d'ordonnancement. Pour résoudre le problème, nous proposons un algorithme dédié, basé sur le calcul de la rétro-propagation des échéances dans le graphe.

2.1.3 Résolution à l'aide de contraintes linéaires

La programmation linéaire [18] est une technique très utilisée pour résoudre des problèmes d'optimisation linéaire, et plus particulièrement d'optimisation combinatoire dans le cadre de la programmation linéaire en nombres entiers. Il s'agit d'un cadre général de résolution de problèmes d'optimisation dans lequel le problème est formulé sous forme d'un programme linéaire, c'est-à-dire un ensemble de contraintes linéaires $C_1,\ldots,C_m$ sur un ensemble fini de variables $z_1,\ldots,z_n$ à valeur dans $\mathbb{R}$ et/ou dans $\mathbb{Z}$ (dans ce cas, il s'agit d'un programme linéaire en nombres entiers), et une fonction linéaire $\phi : \mathbb{R}^n \rightarrow \mathbb{R}$ appelée fonction objectif. Etant données les contraintes linéaires $C_1,\ldots,C_m$ et la fonction objectif ϕ, le but est de trouver une affectation $z_1^*,\ldots,z_n^*$ des variables conforme à leur domaine de définition ($\mathbb{R}^+$ ou $\mathbb{Z}$), telle que les contraintes soient toutes vraies, et telle que $\phi(z_1^*,\ldots,z_n^*)$ soit minimal parmi les affectations qui respectent le domaine de définition et qui satisfont les contraintes. Si c'est un problème de maximisation d'une fonction linéaire ϕ qui nous intéresse, il suffit d'utiliser pour fonction objectif $\phi' = -\phi$.

La résolution d'un programme linéaire qui ne comporte pas de variables entières peut être faite en temps polynomial [27]. De plus, des outils très performants comme CPLEX [5] ou XPress-MP [6] permettent résoudre des programmes linéaires de taille importante. Lorsqu'une partie ou la totalité des variables sont entières, les techniques de séparation et évaluation (ou *branch and bound* en anglais) permettent de trouver la solution par énumération intelligente des valeurs possibles des variables

entières. Ces techniques sont relativement efficace même si elles ne sont pas en temps polynomial. Les problèmes d'ordonnancement s'expriment malheureusement avec des variables entières, ce que nous allons voir dans le cas particulier du problème d'ordonnancement 2.1. Il est donc intéressant de développer des algorithmes spécifiques aux problèmes d'ordonnancement. Nous verrons par exemple dans la section 2.1.4 que le problème d'ordonnancement 2.1 peut être résolu en temps polynomial.

Contraintes linéaires proposées

Soit (G,C,D) un système simple, où $G=(A,\prec)$. Nous allons donner dans la suite un ensemble de contraintes linéaires dont les solutions correspondent aux ordonnancements réalisables de (G,C,D). Tout d'abord, nous supposons que $n=|A|$ et $A=\{\ a_1,\ldots,a_n\ \}$. Sans perdre de généralité, nous supposerons également dans ce qui suit que les durées d'exécution sont strictement positives, c'est-à-dire $C>0$.

Les variables libres des contraintes linéaires présentées ci-dessous sont de deux types. Les variables s_i, $i\in\{1,\ldots,n\}$ à valeur dans $\mathbb{R}^+$ (relation 2.5), représentent les dates de démarrage des actions. Les variables $z_{i,j}$, $(i,j)\in\{1,\ldots,n\}\times\{1,\ldots,n\}$, $i\neq j$, à valeur dans $\{0,1\}$, codent les choix d'ordonnancement. La variable $z_{i,j}$ vaut 1 si et seulement si a_i est ordonnancée avant a_j. Puisque nous considérons des ordonnancements sur une machine monoprocesseur, pour tout $i\neq j$ nous avons l'une au l'autre des deux situations suivantes :

– soit $z_{i,j}=1$ et $z_{j,i}=0$, qui correspond au cas où a_i est ordonnancée avant a_j ;
– soit $z_{i,j}=0$ et $z_{j,i}=1$, qui correspond au cas où a_j est ordonnancée avant a_i.

Dans la suite, M représente un entier suffisamment grand, par exemple $M=\sum_{1\leq i\leq n} C(a_i)$.

$$z_{i,j}+z_{j,i} = 1 \qquad \forall(i,j)\in\{1,\ldots,n\}\times\{1,\ldots,n\}\ .\ i\neq j \tag{2.1}$$

$$z_{i,j} = 1 \qquad \forall(i,j)\in\{1,\ldots,n\}\times\{1,\ldots,n\}\ .\ a_i\prec a_j \tag{2.2}$$

$$s_j-s_i-C(a_i) \geq M(z_{i,j}-1) \qquad \forall(i,j)\in\{1,\ldots,n\}\times\{1,\ldots,n\}\ .\ i\neq j \tag{2.3}$$

$$D(a_i)-s_i-C(a_i) \geq 0 \qquad \forall i\in\{1,\ldots,n\} \tag{2.4}$$

$$s_i \in \mathbb{R}^+ \qquad \forall i\in\{1,\ldots,n\} \tag{2.5}$$

$$z_{i,j} \in \{\ 0,1\ \} \qquad \forall(i,j)\in\{1,\ldots,n\}\times\{1,\ldots,n\}\ .\ i\neq j. \tag{2.6}$$

L'inéquation (2.1) exprime le fait que nous ordonnançons sur une machine monoprocesseur. Ainsi, nous avons forcément soit $z_{i,j}=1$ et $z_{j,i}=0$, soit $z_{i,j}=0$ et $z_{j,i}=1$.

L'inéquation (2.2) traduit les contraintes liées au graphe de précédence. Ainsi, lorsque $a_i\prec a_j$, nous avons forcément a_i qui doit être ordonnancée avant a_j. Nous imposons dans ce cas la valeur de $z_{i,j}$: $z_{i,j}=1$. La valeur de $z_{j,i}$ est dans ce cas également imposée du fait de la présence de l'équation (2.1) : $z_{j,i}=0$.

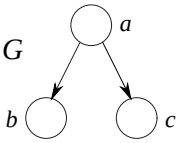
L'inéquation (2.3) traduit l'ordre des actions en contraintes sur les dates de démarrage, c'est-à-dire traduit les valeurs des variables $z_{i,j}$ en inéquations sur les valeurs des variables s_i. Ainsi, lorsque

$z_{i,j} = 1$, nous imposons que la date de terminaison de l'action a_i, c'est-à-dire $s_i + C(a_i)$, soit inférieure à la date de démarrage de l'action a_j, c'est-à-dire s_j.

La relation (2.5) exprime simplement le fait que l'ordonnancement ne peut pas démarrer avant la date 0. Ainsi, nous imposons aux dates de démarrage s_i d'être supérieures ou égales à 0. L'inéquation (2.4) traduit le fait qu'une action doit terminer avant son échéance, c'est-à-dire $D(a_i) \geq s_i + C(a_i)$ pour tout i.

L'ensemble de contraintes linéaires ainsi obtenu comporte des variables à valeurs entières : $z_{i,j}$. Nous allons voir dans le suite comment calculer en temps polynomial des ordonnancements réalisables.

graphe G



action x	$C(x)$	$D(x)$
a	1	5
b	1	5
c	1	5

FIG. 2.4: Exemple de système paramétré.

EXEMPLE 2.4 *Considérons le système simple $(G = (A, \prec), C, D)$, composé de trois actions $A = \{a, b, c\}$, et dont le graphe de précédence G, la fonction de durée d'exécution C et la fonction d'échéance sont données dans la figure 2.4. Nous allons construire l'ensemble des contraintes linéaires qui correspondent au problème d'ordonnancement de ce système simple (problème 2.1).*

$$z_{a,b} + z_{b,a} = 1 \quad (2.7)$$
$$z_{a,c} + z_{c,a} = 1 \quad (2.8)$$
$$z_{b,c} + z_{c,b} = 1 \quad (2.9)$$
$$z_{a,b} = 1 \quad (2.10)$$
$$z_{a,c} = 1 \quad (2.11)$$
$$s_a - s_b - C(b) \geq M(1 - z_{b,a}) \quad (2.12)$$
$$s_b - s_a - C(a) \geq M(1 - z_{a,b}) \quad (2.13)$$
$$s_a - s_c - C(c) \geq M(1 - z_{c,a}) \quad (2.14)$$
$$s_c - s_a - C(a) \geq M(1 - z_{a,c}) \quad (2.15)$$
$$s_b - s_c - C(c) \geq M(1 - z_{c,b}) \quad (2.16)$$
$$s_c - s_b - C(b) \geq M(1 - z_{b,c}) \quad (2.17)$$

$$D(a) - s_a - C(a) \geq 0 \quad (2.18)$$
$$D(b) - s_b - C(b) \geq 0 \quad (2.19)$$
$$D(c) - s_c - C(c) \geq 0 \quad (2.20)$$

$$s_a, s_b, s_c \in \mathbb{R}^+ \qquad \forall i \in \{1,\ldots,n\} \tag{2.21}$$
$$z_{a,b}, z_{b,a}, z_{a,c}, z_{c,a}, z_{b,c}, z_{c,b} \in \{\,0,1\,\}. \tag{2.22}$$

Correspondance entre le problème initial et les contraintes linéaires

Nous allons voir pourquoi l'ensemble des contraintes linéaires est une expression du problème d'ordonnancement 2.1. Pour ce faire, nous montrons qu'à partir de toute solution de cet ensemble de contraintes, nous pouvons construire une solution du problème d'ordonnancement, et réciproquement.

LEMME 2.1 *Lorsque la fonction de durée d'exécution C est strictement positive, les dates de démarrage s_i des actions d'une solution de l'ensemble des contraintes linéaires sont distinctes.*

Preuve : Soit $z_{i,j}$ et s_i une affectation des variables vérifiant l'ensemble de contraintes linéaires. Soient deux indices i et j distincts. Grâce aux équations (2.1) et (2.6), nous pouvons déduire que nous avons soit $z_{i,j} = 1$ et $z_{j,i} = 0$, soit $z_{j,i} = 1$ et $z_{i,j} = 0$. L'équation (2.3) appliquée à $z_{i,j}$ si $z_{i,j} = 1$, ou $z_{j,i}$ si $z_{j,i} = 1$ nous donne soit $s_j - s_i - C(a_i) \geq 0$, soit $s_i - s_j - C(a_j) \geq 0$. Puisque $C > 0$, nous obtenons soit $s_j > s_i$, soit $s_i > s_j$. Nous avons donc dans tous les cas $s_i \neq s_j$. □

Soient $z_{i,j}$ et s_i une affectation des variables vérifiant les contraintes linéaires. Nous allons construire l'ordonnancement α associé, qui sera un ordonnancement réalisable de (G,C,D). Considérons une séquence d'actions α dans laquelle les actions sont ordonnées selon les dates de démarrage s_i des actions. Le lemme 2.1 démontre l'unicité d'une telle séquence d'actions. Nous pouvons définir de façon plus formelle la séquence α, par la suite récurrente suivante :

$$\begin{aligned} \alpha(1) &= a_{i_1} \text{ tel que } s_{i_1} = \mathbf{min}_{1\leq j\leq n}\, s_j \\ \alpha(k) &= a_{i_k} \text{ tel que } s_{i_k} = \mathbf{min}_{a_j \notin \mathsf{ens}(^{k-1}\alpha)}\, s_j. \end{aligned}$$

LEMME 2.2 *L'ordonnancement α ainsi défini est tel que pour tout $k < k'$ nous avons $z_{i_k,i'_k} = 1$.*

Preuve : Soient k et k' deux indices tels que $k < k'$. D'après les équations (2.1) et (2.6), nous avons soit $z_{i_k,i_{k'}} = 1$, soit $z_{i_{k'},i_k} = 1$. Si nous avions $z_{i_{k'},i_k} = 1$, alors l'équation (2.3) nous donnerait $s_{i_k} > s_{i_{k'}}$. Or d'après la définition des indices i_k et i'_k, nous savons que $s_{i_k} < s_{i'_k}$. Nous avons donc forcément $z_{i_k,i_{k'}} = 1$. □

Il nous reste alors à vérifier que la séquence α ainsi définie est bien un ordonnancement réalisable de (G,C,D). Tout d'abord, montrons que $\mathsf{ens}(^k\alpha)$ est fermé en arrière. Soient $k \in \{1,\ldots,|\alpha|\}$ et $\alpha(k) = a_{i_k}$. Soit $a_{i_{k'}} = \alpha(k')$ telle que $a_{i_{k'}} \prec a_{i_k}$. Nous avons donc, d'après l'équation 2.2, $z_{i_{k'},i_k} = 1$. Avec l'équation 2.3, nous obtenons :

$$s_{i_k} - s_{i_{k'}} - C(a_{i_k}) \geq 0.$$

Puisque nous avons supposé $C > 0$, nous en déduisons $s_{i_k} > s_{i_{k'}}$, c'est-à-dire $k > k'$ du fait de la définition de α. Ceci démontre que α est une trace de (G,C,D). Puisque $|\alpha| = n = |A|$, nous pouvons conclure que α est un ordonnancement de (G,C,D).

Nous allons démontrer pour finir que α est un ordonnancement réalisable de (G,C,D), c'est-à-dire que les échéances de toutes les actions sont respectées. Soit $k \in \{1,\ldots,n\}$. Montrons que $D(\alpha(k)) \geq C(^k\alpha)$.

Si $k = 1$, puisque $s_{i_1} \geq 0$ (équation 2.5), et grâce à l'équation 2.4 nous obtenons $D(a_{i_1}) \geq C(a_{i_1})$. Lorsque $k > 1$, nous utilisons les équations 2.2 afin de démontrer tout d'abord que $s_{i_k} \geq s_{i_{k-1}} + C(^{k-1}\alpha)$. Pour ce faire, écrivons 2.2 pour les couples (i_1,i_2), (i_2,i_3), ..., (i_{k-1},i_k) :

$$\begin{aligned}
s_{i_2} - s_{i_1} - C(a_{i_1}) &\geq M(1 - z_{i_1,i_2}) \\
s_{i_3} - s_{i_2} - C(a_{i_2}) &\geq M(1 - z_{i_2,i_3}) \\
&\ldots \\
s_{i_{k-1}} - s_{i_{k-2}} - C(a_{i_{k-2}}) &\geq M(1 - z_{i_{k-2},i_{k-1}}) \\
s_{i_k} - s_{i_{k-1}} - C(a_{i_{k-1}}) &\geq M(1 - z_{i_{k-1},i_k}).
\end{aligned}$$

A l'aide du lemme 2.2, nous savons que le second membre des inégalités précédentes est nul. En sommant ces inégalités, nous obtenons :

$$s_{i_k} - s_{i_1} - C(^{k-1}\alpha) \geq 0.$$

L'équation 2.4 appliquée à l'indice i_k nous donne $D(a_{i_k}) - s_{i_k} - C(a_{i_k}) \geq 0$. Puisque $s_{i_1} \geq 0$ (équation 2.5), nous pouvons conclure :

$$\begin{aligned}
D(a_{i_k}) - s_{i_1} - C(^{k-1}\alpha) - C(a_{i_k}) &\geq 0 \\
D(\alpha(k)) &\geq C(^k\alpha).
\end{aligned}$$

Nous avons donc montré que l'ordonnancement α associé à une solution de l'ensemble des contraintes linéaires est réalisable.

Réciproquement, si $\alpha = a_{i_1} a_{i_2} \ldots a_{i_n}$ est un ordonnacement réalisable, alors l'affectation des variables suivante est une solution de l'ensemble des contraintes linéaires :

$$\begin{aligned}
s_{i_k} &= C(^k\alpha) \\
z_{i_k,i_{k'}} &= 1 \text{ si } k < k', 0 \text{ sinon.}
\end{aligned}$$

La vérification de la satisfaction des contraintes linéaires est laissée en exercice.

2.1.4 Résolution basée sur EDF

A présent, nous allons résoudre le problème d'ordonnancement d'un système simple (problème 2.1) en utilisant un algorithme d'ordonnancement ad hoc. Ce dernier est basé sur la règle EDF (*Earliest Deadline First*), qui consiste à ordonnancer en priorité les actions dont l'échéance est la plus petite. Nous verrons que la complexité de l'algorithme d'ordonnancement proposé est polynomiale par rapport à la taille du système simple d'entrée.

Dans un premier temps, nous définissons la fonction d'ordonnancement t_s qui permet de déterminer si un ordonnancement est réalisable ou non. Pour un ordonnancement donné α, nous verrons qu'il est possible de simplifier le calcul de $t_s(\alpha)$. Nous définirons ensuite la notion d'ordonnancement EDF. Nous verrons que les ordonnancements EDF sont optimaux (dans un sens qui sera précisé), et nous donnerons un algorithme de calcul d'un ordonnancement EDF.

Fonction d'ordonnancement (t_s)

Nous introduisons ici la notion de fonction d'ordonnancement t_s. Pour un ordonnancement α, la valeur $t_s(\alpha)$ représente sa *marge* dans une exécution des actions sans temps mort, selon l'ordre donné par α, et démarrant à la date 0. Ainsi, si $t_s(\alpha)$ est négatif, l'ordonnancement n'est pas réalisable. Au contraire, si $t_s(\alpha)$ est positif, il est possible de retarder la date de démarrage à $t_s(\alpha)$ à la place de 0, tout en respectant les échéances.

DÉFINITION 2.8 *Soit (G,C,D) un système simple et α un ordonnancement de G. Nous définissons la* **fonction d'ordonnancement** *t_s associée à (G,C,D) de la manière suivante. Pour toute trace α, nous définissons :*

$$t_s(\alpha)(k) = D(^k\alpha) - C(^k\alpha)$$

et :

$$t_s(\alpha) = \mathbf{min}_{1\leq k\leq|\alpha|}\ t_s(\alpha)(k).$$

Le calcul de $t_s(\alpha)$ est basé sur les valeurs $t_s(\alpha)(k)$, pour $k \in \{\ 1,\ldots,|\alpha|\ \}$. La valeur $t_s(\alpha)(k)$ représente la marge de la trace $^k\alpha$ par rapport à l'échéance $D(\alpha(k))$, c'est-à-dire la date au plus tard à laquelle la trace $^k\alpha$ peut s'exécuter tout en respectant l'échéance $D(\alpha(k))$. La valeur $t_s(\alpha)$ doit prendre en compte toutes les échéances $D(\alpha(k))$, $k \in \{\ 1,\ldots,|\alpha|\ \}$. Elle est donc calculée comme la marge minimale par rapport à toutes les échéances $D(\alpha(k))$.

PROPOSITION 2.1 *Soit (G,C,D) un système simple. Un ordonnancement α est réalisable si et seulement si :*

$$t_s(\alpha) \geq 0.$$

Nous notons $\mathsf{real}(G,C,D)$ *l'ensemble des ordonnancements réalisables de (G,C,D).*

Preuve : Soit α un ordonnancement. Par définition, α est réalisable si et seulement si :

$$\begin{aligned}
& \forall k \in \{\ 1,\ldots,|\alpha|\ \}\ .\ D(^k\alpha) \geq C(^k\alpha) \\
\Leftrightarrow\ & \forall k \in \{\ 1,\ldots,|\alpha|\ \}\ .\ D(^k\alpha) - C(^k\alpha) \geq 0 \\
\Leftrightarrow\ & \forall k \in \{\ 1,\ldots,|\alpha|\ \}\ .\ t_s(\alpha)(k) \geq 0 \\
\Leftrightarrow\ & \mathbf{min}_{1\leq k\leq|\alpha|}\ t_s(\alpha)(k)\ \geq\ 0 \\
\Leftrightarrow\ & t_s(\alpha) \geq 0. \quad \square
\end{aligned}$$

La proposition précédente montre comment la fonction d'ordonnancement t_s peut être utilisée afin de déterminer si un ordonnancement α est réalisable ou non. Son calcul est effectué par un minimum entre $|\alpha|$ valeurs. Nous allons voir dans la suite comment réduire le nombre de valeurs intervenant dans ce calcul de minimum.

EXEMPLE 2.5 *Reprenons l'exemple 2.3. Considérons les ordonnancements* $\alpha_1 = abcegdf$ *et* $\alpha_2 = abdfceg$. *Nous avons vu que* α_1 *est réalisable. Nous devons donc avoir* $t_s(\alpha_1) \geq 0$. *Calculons* $t_s(\alpha_1)$ *:*

$$\begin{aligned}
t_s(\alpha_1) &= \mathbf{min}_{1\leq k\leq 7}\, D(^k(abcegdf)) - C(^k(abcegdf)) \\
&= \mathbf{min}\{\, D(a)-C(a), D(b)-C(ab), D(c)-C(abc), D(e)-C(abce), D(g)-C(abceg), \\
&\quad D(d)-C(abcegd), D(f)-C(abcegdf)\,\} \\
&= \mathbf{min}\{\, 1-1, +\infty-3, +\infty-4, +\infty-5, 8-7, 9-9, +\infty-12\,\} \\
&= \mathbf{min}\{\, 0, +\infty, +\infty, +\infty, 1, 0, +\infty\,\} \\
&= 0.
\end{aligned}$$

L'ordonnancement α_2 *n'est pas réalisable. Ainsi, nous devons avoir* $t_s(\alpha_2) < 0$, *ce qui est confirmé par le calcul de* $t_s(\alpha_2)$ *:*

$$\begin{aligned}
t_s(\alpha_2) &= \mathbf{min}_{1\leq k\leq 7}\, D(^k(abdfceg)) - C(^k(abdfceg)) \\
&= \mathbf{min}\{\, D(a)-C(a), D(b)-C(ab), D(d)-C(abd), D(f)-C(abdf), D(c)-C(abdfc), \\
&\quad D(e)-C(abdfce), D(g)-C(abdfceg)\,\} \\
&= \mathbf{min}\{\, 1-1, +\infty-3, 9-5, +\infty-8, +\infty-9, +\infty-10, 8-12\,\} \\
&= \mathbf{min}\{\, 0, +\infty, 3, +\infty, +\infty, +\infty, -4\,\} \\
&= -4.
\end{aligned}$$

Pour un ordonnancement α et des dates d'échéances D données, certaines échéances sont plus difficiles à satisfaire que d'autres. Nous définissons dans ce qui suit le sous-ensemble d'indices $\mathsf{critique}_D(\alpha)$, dans lequel les indices des échéances les plus faciles à satisfaire ont été retirées.

DÉFINITION 2.9 *Soit* (G,C,D) *un système simple et* α *une séquence d'actions. Nous définissons le sous-ensemble d'indices* $\mathsf{critique}_D(\alpha) \subseteq \{\, 1,\ldots,|\alpha|\,\}$ *de la manière suivante :*

$$\mathsf{critique}_D(\alpha) = \{\, k \mid \forall k' > k \,.\, D(^{k'}\alpha) > D(^k\alpha)\,\}.$$

Le calcul de $\mathsf{critique}_D(\alpha)$ est réalisé en éliminant les indices i qui correspondent à des échéances $D(^i\alpha)$ dont nous savons qu'il existe un indice supérieur $j > i$ qui correspond à une échéance $D(^j\alpha)$ plus petite ($D(^j\alpha) < D(^i\alpha)$). En effet, si $j > i$ et $D(^j\alpha) < D(^i\alpha)$, l'échéance $D(^j\alpha)$ est forcément plus difficile à satisfaire que l'échéance $D(^i\alpha)$. Autrement dit, le respect de $D(^j\alpha)$ implique le respect de $D(^i\alpha)$, c'est-à-dire $t_s(\alpha)(j) < t_s(\alpha)(i)$, ce qui motive la proposition suivante.

PROPOSITION 2.2 *Soit* (G,C,D) *un système simple, et* t_s *sa fonction d'ordonnancement. Nous avons alors :*

$$t_s(\alpha) = \mathbf{min}_{1\leq k\leq|\alpha|}\, t_s(\alpha)(k) = \mathbf{min}_{k\in\mathsf{critique}_D(\alpha)}\, t_s(\alpha)(k).$$

Pour démontrer la proposition 2.2, nous démontrons d'abord un résultat intermédiaire concernant les échéances D et le sous-ensemble d'indices $\mathsf{critique}_D(\alpha)$. Ce résultat est donné par le lemme suivant.

LEMME 2.3 *Soit (G,C,D) un système simple et α un ordonnancement de G. Considérons une écriture ordonnée de l'ensemble* $\mathsf{critique}_D(\alpha)$, $\mathsf{critique}_D(\alpha) = \{\ k_1,\ldots,k_m\ \}$, *dans laquelle* $m = |\mathsf{critique}_D(\alpha)|$ *et les valeurs* k_i *vérifient* $k_i < k_{i+1}$ *pour tout* i. *En posant* $k_0 = 0$, *l'ensemble* $\mathsf{critique}_D(\alpha) = \{k_1,\ldots,k_m\}$ *vérifie alors :*

$$\forall k\ .\ k_{i-1} < k < k_i\ \Rightarrow\ D(\alpha(k)) \geq D(\alpha(k_i)).$$

Preuve du lemme 2.3 : Nous démontrons le résultat par l'absurde. Supposons $D(\alpha(k_i)) > D(\alpha(k))$. Soit k' le dernier indice tel que $k \leq k' < k_i$ et :

$$D(\alpha(k')) = \mathbf{min}\ \{\ D(\alpha(k)), D(\alpha(k+1)),\ldots,D(\alpha(k_i-1))\ \}.$$

Remarquons que :

$$D(\alpha(k_i)) > D(\alpha(k)) \geq D(\alpha(k')). \tag{2.23}$$

Nous avons donc :

$$\forall l \in \{k'+1,\ldots,k_i-1\}\ .\ D(\alpha(l)) > D(\alpha(k')). \tag{2.24}$$

Puisque $k_i \in \mathsf{critique}_D(\alpha)$, et par (2.23) nous obtenons :

$$\forall l > k_i\ .\ D(\alpha(l)) > D(\alpha(k_i)) > D(\alpha(k')). \tag{2.25}$$

Ainsi, par (2.24) et (2.25) nous avons :

$$\forall l > k'\ .\ D(\alpha(l)) > D(\alpha(k_i)) > D(\alpha(k')).$$

Ceci démontre que $k' \in \mathsf{critique}_D(\alpha)$. D'où une contradiction. □

Preuve de la proposition 2.2 : Soit $k \in \{1,\ldots,|\alpha|\}$ un indice tel que $k \notin \mathsf{critique}_D(\alpha)$. Nous allons démontrer qu'il existe $k' \in \mathsf{critique}_D(\alpha)$ tel que $t_s(\alpha)(k) \geq t_s(\alpha)(k')$. Puisque $k \notin \mathsf{critique}_D(\alpha)$ et $|\alpha| \in \mathsf{critique}_D(\alpha)$, nous avons $k < |\alpha|$. Soit k' l'indice défini par :

$$k' = \mathbf{min}_{k''>k}\ k'' \in \mathsf{critique}_D(\alpha).$$

Remarquons que $\{\ k' \in \mathsf{critique}_D(\alpha)\ |\ k' > k\ \}$ est non vide puisqu'il contient $|\alpha|$. Nous avons :

$$t_s(\alpha)(k) = D(\alpha(k)) - C(^k\alpha).$$

Comme $k' > k$, nous obtenons : $t_s(\alpha)(k) \geq D(\alpha(k)) - C(^{k'}\alpha)$. D'après le lemme 2.3, nous avons également $D(\alpha(k)) \geq D(\alpha(k'))$. Ceci permet de conclure :

$$t_s(\alpha)(k) \geq D(\alpha(k')) - C(^{k'}\alpha) = t_s(\alpha)(k').\quad \square$$

EXEMPLE 2.6 *Reprenons le système simple (G,C,D) et l'ordonnancement α_2 de l'exemple 2.5. Calculons l'ensemble* $\mathsf{critique}_D(\alpha_2)$ *:*

$$\mathsf{critique}_D(\alpha_2) = \{\ 1,7\ \}.$$

Ainsi, le calcul de $t_s(\alpha_2)$ se ramène simplement à :

$$t_s(\alpha_2) = \mathbf{min}\{\ D(a)-C(a), D(g)-C(abdfceg)\ \} = \mathbf{min}\{\ 1-1, 8-12\ \} = -4.$$

Ordonnancements EDF

Intuitivement, en ordonnançant les actions les plus urgentes en premier, nous pouvons espérer qu'il sera facile de respecter les échéances. Une telle manière d'ordonnancer est dite EDF (*Earliest Deadline First*). Dans un ordonnancement EDF, les actions dont l'échéance sont les plus petites sont ordonnancées en priorité. Les ordonnancements EDF sont couramment employés pour répondre à des problèmes d'ordonnancement dans lesquels les contraintes sont des échéances à respecter [16].

DÉFINITION 2.10 *Soit (G,C,D) un système simple. Nous définissons la fonction d'échéance D^* induite par G et D de la manière suivante :*

$$D^*(a) = \mathbf{min}\{\, D(a') \mid a' = a \;\vee\; a \prec a' \}.$$

Nous dirons alors qu'un ordonnancement α est un ordonnancement **EDF** *par rapport à la fonction d'échéance D si :*

$$i < j \;\Rightarrow\; D^*({}^i\alpha) \leq D^*({}^j\alpha).$$

Nous notons $\mathsf{EDF}(G,D)$ l'ensemble des ordonnancements EDF du graphe G et la fonction d'échéance D.

De même, nous dirons qu'un prolongement α' d'une trace α dans le graphe G est **EDF** *par rapport à la fonction d'échéance D si α' est un ordonnancement EDF du graphe résiduel de G après l'exécution de α.*

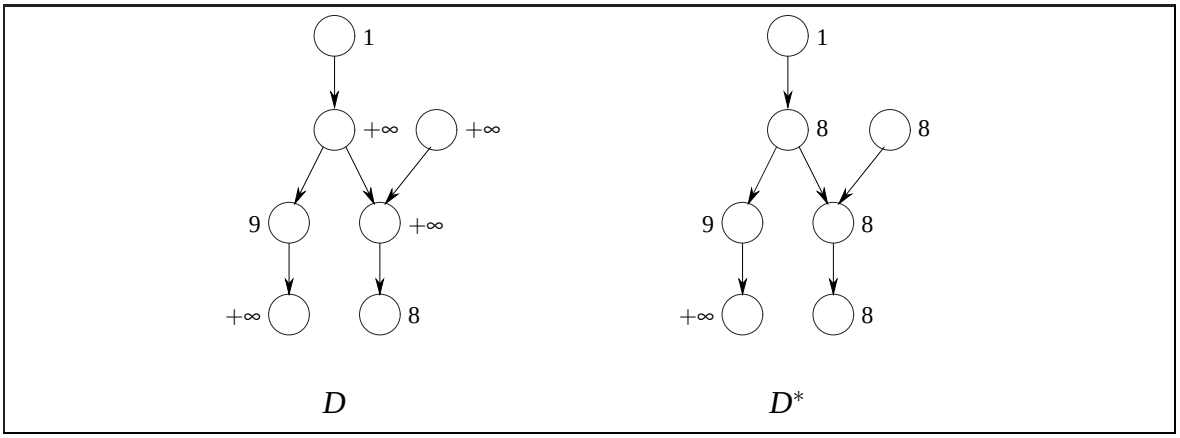


FIG. 2.5: Exemple de rétro-propagation des échéances dans un graphe.

La fonction d'échéance D^* est issue d'une "rétro-propagation" des échéances initiales. Pour une action a donnée, la valeur $D^*(a)$ représente une synthèse des échéances de toutes les actions qui dépendent de a. Afin de prendre en compte la contrainte la plus forte, cette synthèse est réalisée par un minimum. La fonction D^* peut être utilisée pour définir des priorités inverses sur les actions de A : une action a est plus prioritaire qu'une action b si $D^*(a) < D^*(b)$. Un ordonnancement obtenu en appliquant de telles priorités est appelé ordonnancement EDF. Nous verrons dans la suite que les ordonnancements EDF sont optimaux, dans un sens qui sera précisé.

La figure 2.5 reprend l'exemple de la figure 2.2. Elle fournit une comparaison entre les échéances initiales D et les échéances rétro-propagées D^*. Un ordonnancement EDF du graphe est donné, par exemple, par $\alpha = abcegdf$.

PROPOSITION 2.3 *Soit (G,C,D) un système simple, D^* la fonction d'échéance induite par G et D. Alors l'ordre $\prec_{EDF}$ est compatible avec l'ordre initial $\prec$ du graphe de précédence $G = (A, \prec)$, c'est-à-dire :*

$$a \prec_{EDF} b \;\Rightarrow\; b \not\prec a.$$

DÉFINITION 2.11 *Soit (G,C,D) un système simple. Considérons la relation d'équivalence $\equiv_{D^*}$ sur les actions A définie par :*

$$a \equiv_{D^*} b \iff D^*(a) = D^*(b).$$

Nous dirons que la partition $A_1 \ldots A_L$ de A induite par la relation d'équivalence $\equiv_{D^*}$ est la **partition induite par** *D^*. Nous appellerons* **classe EDF** *un élément A_i de la partition.*

La fonction D^* est constante sur une classe EDF A_i. Nous noterons $D^*(A_i)$ la valeur prise par D^* sur A_i, c'est-à-dire :

$$\forall a \in A_i \;.\; D^*(a) = D^*(A_i).$$

La fonction D^* induit une partition de l'ensemble des actions A en classes dont les actions ont même niveau de priorité vis à vis d'EDF. Nous allons voir dans la suite quelques propriétés élémentaires sur les ordonnancement EDF. Ces dernières découlent directement des définitions des classes EDF et des ordonnancements EDF.

EXEMPLE 2.7 *Reprenons le système simple (G,C,D) de l'exemple 2.2. La fonction D^* associée à D est donnée par la figure 2.5. Elle induit la partition $A_1 = \{\ a\ \}$, $A_2 = \{\ b,c,e,g\ \}$, $A_3 = \{\ d\ \}$, $A_4 = \{\ f\ \}$ de l'ensemble des actions A. Celle-ci est représentée dans la figure 2.6.*

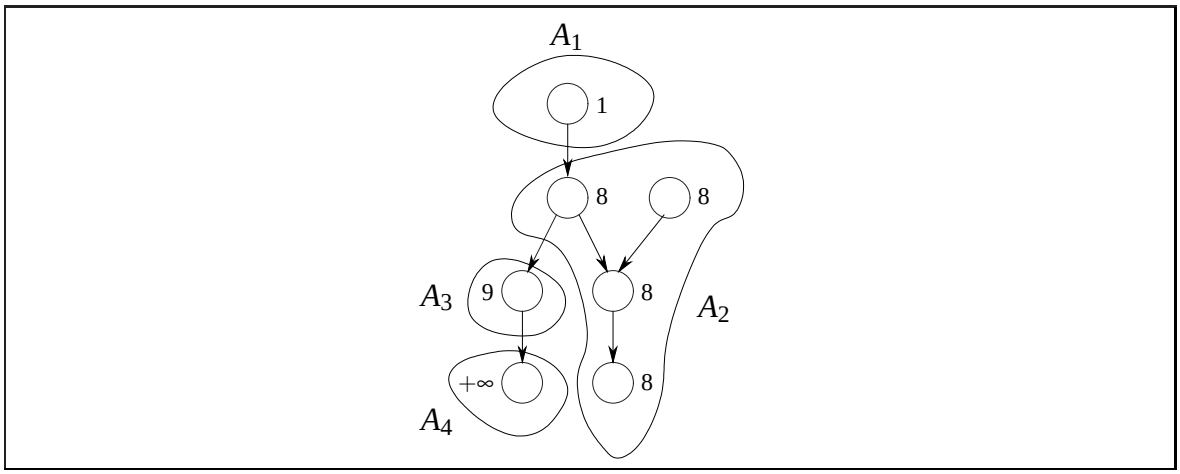


FIG. 2.6: Partitionnement du graphe selon les échéances propagées.

PROPOSITION 2.4 *Soit (G,C,G) un système simple. Considérons la partition $A_1 \ldots A_L$ induite par D^* telle que celle-ci soit écrite par échéances croissantes, c'est-à-dire $D^*(A_1) < \ldots < D^*(A_L)$. Nous avons les résultats suivants.*

1. *L'ensemble $\mathsf{EDF}(G,D)$ des ordonnancements EDF de G est aussi l'ensemble de tous les ordonnancements du graphe $(G/A_1)(G/A_2)\ldots(G/A_L)$.*
2. *N'importe quel ordonnancement EDF α peut s'écrire $\alpha = \alpha_1 \ldots \alpha_L$, où pour tout l, α_l est un ordonnancement de G/A_l.*
3. *Pour un ordonnancement EDF α, l'ensemble $\mathsf{critique}_D(\alpha)$ peut s'écrire $\mathsf{critique}_D(\alpha) = \{\ |A_1|, |A_1 \cup A_2|, \ldots, |A_1 \cup A_2 \cup \ldots \cup A_L|\ \}$.*

Preuve de la proposition : Les deux premiers points découlent directement des définitions 2.10 et 2.11.

3. Soit α un ordonnacement EDF. D'après le 2, tout ordonnancement EDF α s'écrit $\alpha = \alpha_1 \ldots \alpha_L$, où α_l est un ordonnancement de la classe A_l, et $A_1, \ldots, A_L$ est une partition ordonnée des actions A, c'est-à-dire $D(A_1) < D(A_2) < \ldots < D(A_L)$.

Soit k un indice de la forme $k = |\alpha_1 \ldots \alpha_l|$, $l \in \{\ 1, \ldots, L\ \}$. Nous avons $D(\alpha(k)) = D(A_l)$, et pour tout $k' > k$, nous avons $D(\alpha(k')) \in \{\ D(A_{l+1}), D(A_{l+2}), \ldots, D(A_L)\ \}$. La partition étant ordonnée, nous en déduisons que $D(\alpha(k)) < D(\alpha(k'))$, c'est-à-dire que $k \in \mathsf{critique}_D(\alpha)$. Comme k est un indice quelconque de la forme $k = |\alpha_1 \ldots \alpha_l| = |A_1 \ldots A_l|$, nous avons démontré $\{\ |A_1|, |A_1A_2|, \ldots, |A_1A_2 \ldots A_L|\ \} \subseteq \mathsf{critique}_D(\alpha)$. L'égalité s'obtient en remarquant pour pour tout indice k tel que $|\alpha_1 \ldots \alpha_{l-1}| < k < |\alpha_1 \ldots \alpha_l|$. Il existe donc $k' = |\alpha_1 \ldots \alpha_l|$. tel que $k' > k$ et $D(\alpha(k')) = D(\alpha(k)$, c'est-à-dire $k \notin \mathsf{critique}_D(\alpha)$. Ceci termine la démonstration de $\{\ |A_1|, |A_1A_2|, \ldots, |A_1A_2 \ldots A_L|\ \} = \mathsf{critique}_D(\alpha)$ □

EXEMPLE 2.8 *Reprenons le système simple (G,C,D) de l'exemple 2.2. La fonction d'échéance D^* induit la partition $A_1 = \{\ a\ \}$, $A_2 = \{\ b,c,e,g\ \}$, $A_3 = \{\ d\ \}$, $A_4 = \{\ f\ \}$ de l'ensemble des actions A est telle que $D^*(A_1) < D^*(A_2) < D^*(A_3) < D^*(A_4)$. Ainsi, les ordonnancement α EDF de G par rapport à D sont de la forme $\alpha = a\alpha' d f$, où α' est un ordonnancement de G/A_2, c'est-à-dire $\alpha' = bceg$ ou $\alpha' = cbeg$. Ainsi, les ordonnancements EDF de G par rapport à D sont soit $\alpha_1 = abcegd f$, soit $\alpha_2 = acbegd f$.*

PROPOSITION 2.5 *Soit (G,C,D) un système simple et α un ordonnancement. Nous avons les résultats suivants.*

1. *$D^* \leq D$.*
2. *$\mathsf{critique}_D(\alpha) = \mathsf{critique}_{D^*}(\alpha)$.*
3. *Pour tout $k \in \mathsf{critique}_D(\alpha)$, $D(\alpha(k)) = D^*(\alpha(k))$.*

Afin de simplifier la démonstration de la proposition 2.5, nous démontrons d'abord un résultat concernant les fonctions d'échéance D et D^*. Celui-ci montre que le minimum des fonctions D et D^* sont identiques sur un suffixe de α.

LEMME 2.4 *Soit (G,C,D) un système simple et α un ordonnancement. Pour tout $k \in \{ 1,\ldots,|\alpha| \}$, nous avons :*

$$\mathbf{min}_{k' \geq k}\, D(\alpha(k')) = \mathbf{min}_{k' \geq k}\, D^*(\alpha(k')).$$

Preuve du lemme : Puisque $D^* \leq D$, nous avons $\mathbf{min}_{l \geq k}\, D^*(\alpha(l)) \leq \mathbf{min}_{l \geq k}\, D(\alpha(l))$. Soit $k' \geq k$ tel que $D(\alpha(k')) = \mathbf{min}_{l \geq k}\, D(\alpha(l))$. Soit $a = \alpha(k')$. L'indice k' vérifie alors :

$$D^*(\alpha(k')) = D^*(a) = \mathbf{min}_{a \prec a'}\, D(a').$$

Puisque α est un ordonnancement de G, pour tout a' tel que $a \prec a'$, il existe $l \geq k'$ tel que $a' = \alpha(l)$. Nous avons donc :

$$D^*(\alpha(k')) \geq \mathbf{min} \{ D(\alpha(l)) \mid l \geq k \}.$$

Puisque $k' \geq k$, nous obtenons $\mathbf{min} \{ D^*(\alpha(l)) \mid l \geq k \} \geq \mathbf{min} \{ D(\alpha(l)) \mid l \geq k \}$. □

Preuve de la proposition 2.5 : 1. Soit $a \in A$ une action. Comme $a \in \{a' \mid a' = a \vee a \prec a' \}$, nous avons :

$$D^*(a) = \mathbf{min}_{a'=a \,\vee\, a \prec a'}\, D(a') \;\leq\; D(a).$$

Ceci démontre que $D^* \leq D$.

3. Tout d'abord, nous allons démontrer que pour tout $k \in \mathsf{critique}_D(\alpha)$, $D(\alpha(k)) = D^*(\alpha(k))$. Soit $k \in \mathsf{critique}_D(\alpha)$. Puisque $k \in \mathsf{critique}_D(\alpha)$ nous avons, pour tout $k' > k$, $D(\alpha(k')) > D(\alpha(k))$. Nous en déduisons $D(\alpha(k)) = \mathbf{min} \{ D(\alpha(k')) \mid k' \geq k \}$. Par le lemme 2.4, nous obtenons :

$$D(\alpha(k)) = \mathbf{min} \{ D^*(\alpha(k')) \mid k' \geq k \} \leq D^*(\alpha(k)).$$

D'après le 1 de la proposition, nous avons $D^* \leq D$. Nous pouvons ainsi conclure que $D(\alpha(k)) = D^*(\alpha(k))$.

2. Démontrons d'abord que $\mathsf{critique}_{D^*}(\alpha) \subseteq \mathsf{critique}_D(\alpha)$. Soit $k \in \mathsf{critique}_{D^*}(\alpha)$. Nous avons, pour tout $k' > k$:

$$D^*(\alpha(k)) < D^*(\alpha(k')).$$

Puisque $D^* \leq D$, nous avons en particulier $D^*(\alpha(k')) \leq D(\alpha(k'))$. De plus, puisque $k \in \mathsf{critique}_{D^*}(\alpha)$, nous avons $D(\alpha(k)) = D^*(\alpha(k))$ (3 de la proposition). Ainsi $D(\alpha(k)) < D(\alpha(k'))$, c'est-à-dire $k \in \mathsf{critique}_D(\alpha)$.

Démontrons que $\mathsf{critique}_D(\alpha) \subseteq \mathsf{critique}_{D^*}(\alpha)$. Soit $k \in \mathsf{critique}_D(\alpha)$. Puisque $k \in \mathsf{critique}_D(\alpha)$, nous avons :

$$D^*(\alpha(k)) = D(\alpha(k)) < \mathbf{min} \{ D(\alpha(k')) \mid k' > k \}.$$

En utilisant le lemme 2.4, nous obtenons $D^*(\alpha(k)) < \mathbf{min} \{ D^*(\alpha(k')) \mid k' > k \}$, c'est-à-dire $k \in \mathsf{critique}_{D^*}(\alpha)$. Nous avons ainsi démontré que $\mathsf{critique}_D(\alpha) \subseteq \mathsf{critique}_{D^*}(\alpha)$. □

PROPOSITION 2.6 *Soit (G,C,D) un système simple et t_s la fonction d'ordonnancement qui lui est associée. Alors la fonction t_s^* associée à (G,C,D^*) est telle que, pour toute ordonnancement α :*

$$t_s(\alpha) = t_s^*(\alpha).$$

Nous avons donc $\mathsf{real}(G,C,D) = \mathsf{real}(G,C,D^*)$.

Preuve de la proposition : La résultat découle directement de ceux introduits dans le proposition 2.5. En effet, nous avons :

$$\begin{aligned} t_s(\alpha) &= \mathbf{min}_{k\in \mathsf{critique}_{D}(\alpha)} D(\alpha^k) - C(\alpha^k) = \mathbf{min}_{k\in \mathsf{critique}_{D}(\alpha)} D(\alpha(k)) - C(\alpha^k) \text{ et} \\ t_s^*(\alpha) &= \mathbf{min}_{k\in \mathsf{critique}_{D^*}(\alpha)} D^*(\alpha^k) - C(\alpha^k) = \mathbf{min}_{k\in \mathsf{critique}_{D^*}(\alpha)} D^*(\alpha(k)) - C(\alpha^k). \end{aligned}$$

D'après la proposition 2.5 nous avons $\mathsf{critique}_D(\alpha) = \mathsf{critique}_{D^*}(\alpha)$, et pour tout $k \in \mathsf{critique}_D(\alpha)$ nous avons $D(\alpha(k)) = D^*(\alpha(k))$, nous pouvons conclure :

$$t_s^*(\alpha) = \mathbf{min}_{k\in \mathsf{critique}_{D}(\alpha)} D(\alpha(k)) - C(\alpha^k) = t_s(\alpha). \quad \square$$

PROPOSITION 2.7 *Soit (G,C,D) un système simple. Alors les ordonnancements EDF maximisent la fonction t_s, c'est-à-dire qu'un ordonnancement EDF α_{EDF} est tel que pour tout ordonnancement α nous avons :*

$$t_s(\alpha_{EDF}) \geq t_s(\alpha).$$

La preuve de la proposition est basée sur le lemme suivant. Il s'agit d'un résultat sur l'échange de deux actions indépendantes et consécutives d'un ordonnancement, lorsque les échéances sont inversées.

LEMME 2.5 *Soit α un ordonnancement tel qu'il existe deux actions consécutives et indépendantes $a = \alpha(i)$ et $b = \alpha(i+1)$, telles que leurs échéances sont inversées, c'est-à-dire $D(a) \geq D(b)$. Considérons l'ordonnancement α' dans lequel a et b sont échangées, c'est-à-dire $\alpha'(j) = \alpha(j)$ pour tout $j \notin \{i, i+1\}$, $\alpha'(i) = b$ et $\alpha'(i+1) = a$. Alors :*

$$t_s(\alpha') \geq t_s(\alpha).$$

Preuve du lemme : Nous avons :

$$\begin{aligned} t_s(\alpha)(i) &= D(a) - C(^{i-1}\alpha) - C(a) \\ t_s(\alpha)(i+1) &= D(b) - C(^{i-1}\alpha) - C(a) - C(b). \end{aligned}$$

Nous en déduisons que :

$$t_s(\alpha)(i) \geq t_s(\alpha)(i+1). \tag{2.26}$$

Nous avons $t_s(\alpha')(j) = t_s(\alpha)(j)$ pour tout j, sauf pour $j \in \{i, i+1\}$:

$$\begin{aligned} t_s(\alpha')(i) &= D(b) - C(^{i-1}\alpha) - C(b) \\ t_s(\alpha')(i+1) &= D(a) - C(^{i-1}\alpha) - C(a) - C(b). \end{aligned}$$

Ainsi, nous avons :

$$t_s(\alpha')(i) \geq t_s(\alpha)(i+1) \tag{2.27}$$

$$t_s(\alpha')(i+1) \geq t_s(\alpha)(i+1). \tag{2.28}$$

Nous déduisons ainsi de (2.26),(2.27) et (2.28), que :

$$\begin{array}{rcl} \mathbf{min}_{1\leq j\leq|\alpha'|}\, t_s(\alpha')(j) & \geq & \mathbf{min}_{1\leq j\leq|\alpha|}\, t_s(\alpha)(j) \\ t_s(\alpha') & \geq & t_s(\alpha). \quad \square \end{array}$$

Preuve de la proposition 2.7 : Nous appliquons le lemme 2.5 comme suit. Soit α un ordonnancement quelconque, et α_{EDF} un ordonnancement EDF. Il est possible d'obtenir α_{EDF} à partir de α en commutant successivement des actions consécutives ayant des échéances D^* inversées. $\square$

La proposition 2.7 montre que les ordonnancements EDF sont ceux qui maximisent la fonction t_s. Ainsi, s'il existe un ordonnancement réalisable, n'importe quel ordonnancement EDF sera réalisable. Les ordonnancements EDF répondent donc au problème de recherche d'ordonnancements réalisables d'un système simple (G,C,D). De plus, nous allons voir que le calcul d'un ordonnancement EDF peut être fait en temps polynomial.

EXEMPLE 2.9 *Reprenons le système simple (G,C,D) défini dans l'exemple 2.2. Nous avons vu dans l'exemple 2.8 que les ordonnancements EDF de G par rapport à D sont soit $\alpha_1 = abcegdf$, soit $\alpha_2 = acbegdf$. D'après la proposition 2.5, nous avons :*

$$\mathsf{critique}_D(\alpha_1) = \mathsf{critique}_D(\alpha_2) = \{\ 1,5,6,7\ \}.$$

Ainsi, nous avons :

$$t_s(\alpha_1) = t_s(\alpha_2) = \mathbf{min}\{\ 1-1, 8-7, 9-9, +\infty-12\ \} = 0.$$

Or la fonction d'ordonnancement t_s est telle que $t_s \leq 0$. En effet, puisque $D(a_1) = 1$ et $C(a_1) = 1$, nous avons pour tout ordonnancement α :

$$t_s(\alpha) \leq t_s(\alpha)(k) = D(a_1) - C(\alpha^k) \leq D(a_1) - C(a_1) = 0$$

où k est la position de a_1 dans l'ordonnancement α, c'est-à-dire $\alpha(k) = a_1$.

Ainsi, les ordonnancements EDF maximisent la fonction d'ordonnancement t_s pour ce système paramétré (G,C,D) particulier.

Algorithme d'ordonnancement proposé

Nous proposons ici un algorithme d'ordonnancement qui résout le problème de l'ordonnancement d'un système simple (G,C,D) par le calcul d'un ordonnancement EDF. Etant donné un système (G,C,D), l'algorithme procède d'abord à une rétro-propagation des échéances, c'est-à-dire au calcul de la fonction D^*. Les actions sont ensuite ordonnées en respectant les contraintes de précédence imposées par G et l'ordre induit par D^*.

Nous donnons d'abord l'algorithme de calcul de D^*, appelé *RetroEcheances* (figure 2.7). Il s'agit essentiellement d'une boucle dont le nombre d'itérations est égal au nombre d'actions du système simple (G,C,D). Le sous-ensemble d'actions R correspond aux actions qui n'ont pas encore été

traitées. Celui-ci est initialisé à A au démarrage de l'algorithme. Le sous-ensemble S de R contient les actions dont tous les successeurs ont déjà été traités. A chaque itération, le corps de la boucle calcule la valeur $D^*(a)$ d'une action a de S. L'ensemble S est re-calculé à chaque fois, afin de simplifier l'écriture de l'algorithme. Cependant, il est possible d'éviter ce calcul complet en mettant à jour S à chaque fois qu'une action a est traitée.

L'algorithme *RetroEcheances* a une durée d'exécution polynomial en la taille de l'instance d'entrée. En effet, il est constitué d'une boucle principale dont le nombre d'itérations est égal au nombre d'actions $|A|$. De plus, les calculs effectués dans le corps de la boucle, c'est-à-dire le calcul d'un minimum et des successeurs des actions non encore traitées, sont de durée polynomiale.

RetroEcheances$(G = (A, \prec), D)$
  $R := A$
  tant que $(R \neq \emptyset)$ **faire**
    $S := \{\ a \in R \mid \mathsf{succ}_{G/R}(a) = \emptyset\ \}$
    choisir $a \in S$
    $D^*(a) :=$ **min** $\{\ D(a)\ \} \cup \{\ D^*(a') \mid a' \in \mathsf{succ}_G(a)\ \}$
    $R := R \setminus \{a\}$
  fin tant que
  retourner D^*
fin *RetroEcheance*

FIG. 2.7: Algorithme de calcul de la fonction D^*.

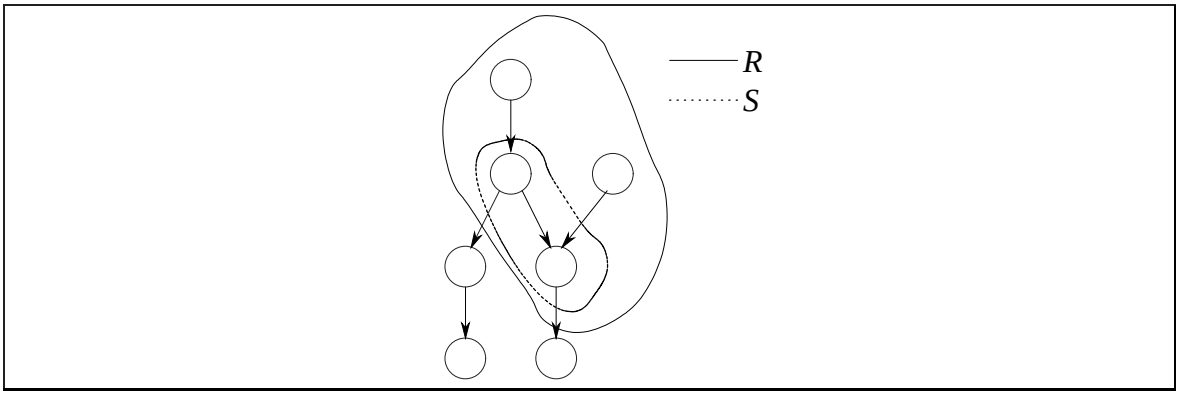


FIG. 2.8: Exemple d'ensembles R et S considérés par l'algorithme *RetroEcheances*.

Nous donnons ensuite l'algorithme d'ordonnancement d'un système simple (G, C, D) dans la figure 2.9. Puisque l'algorithme calcule un ordonnancement EDF, il n'utilisera pas la fonction C. L'algorithme calcule d'abord la fonction D^* par un appel à *RetroEcheances* décrit ci-dessus. Ensuite, une boucle permet de construire un ordonnancement EDF. Celle-ci est itérée autant de fois que le système comporte d'actions. Pour finir, l'algorithme calcule $t_s(\alpha)$ afin de savoir si l'ordonnancement α EDF ainsi calculé est réalisable. Si α est réalisable, la fonction *OrdonnancerEDF* retourne l'ordonnancement α. Sinon, la fonction retourne $\emptyset$, ce qui signifie qu'il n'existe pas d'ordonnancement réalisable

pour le système simple (G,C,D) donné en entrée.

OrdonnancerEDF $(G = (A, \prec), C, D)$
  $D^*(a) := RetroEcheances(G, D)$
  $\alpha := \varepsilon$
  $R := A$
  tant que $(R \neq \emptyset)$ **faire**
    $S := \{\ a \in R \mid \mathsf{pred}_{G/R}(a) = \emptyset\ \}$
    choisir $a \in S \mid D^*(a) = \mathbf{min}_{a' \in S}\, D^*(a')$
    $\alpha := \alpha a$
    $R := R \setminus \{a\}$
  fin tant que
  si $(t_s(\alpha) \geq 0)$ **faire retourner** α
  sinon faire retourner $\emptyset$
fin *OrdonnancerEDF*

FIG. 2.9: Algorithme de calcul d'un ordonnancement EDF.

2.2 Contrôle de qualité sans incertitude

2.2.1 Définition du problème

Dans cette section, nous considérons des systèmes dans lesquels les durées d'exécution dépendent directement d'un paramètre entier, appelé paramètre de *qualité* $q \in Q$. Le problème d'ordonnancement de la section 2.1.2 est un sous problème du problème que nous présentons ici, et correspond au cas où l'ensemble des niveaux de qualité Q est tel que $|Q| = 1$, c'est-à-dire qu'il n'y a pas véritablement de choix concernant les niveaux de qualité.

Dans cette section, nous considérons des durées d'exécution paramétrées par les niveaux de qualité, c'est-à-dire des fonctions C de la forme $C : A \times Q \to \mathbb{R}^+$. L'objectif est alors non seulement de respecter les échéances D, mais également de maximiser la qualité globale, dans un sens qui sera précisé plus tard. Le problème d'ordonnancement comporte donc deux dimensions de choix : le choix de l'ordre, et celui des niveaux de qualité. Nous verrons dans la suite que les choix de niveaux de qualité rendent le problème NP-complet.

DÉFINITION 2.12 *Nous dirons que le quadruplet* (G,C,D,Q) *est un* **système paramétré** *si :*

1. $G = (A, \prec)$ *est un graphe de précédence.*
2. $Q \subseteq \mathbb{N}$ *est un ensemble fini de niveaux de qualité. Nous noterons* $q_{min} = \mathbf{min}_{q \in Q}\ q$ *et* $q_{max} = \mathbf{max}_{q \in Q}\ q$*.*
3. D *est une fonction d'échéance, c'est-à-dire une fonction de la forme* $D : A \to \mathbb{R}^+ \cup \{\ +\infty\ \}$*.*
4. C *est une fonction de durée d'exécution paramétrée, c'est-à-dire une fonction de la forme* $C : A \times Q \to \mathbb{R}^+$*, telle que pour tout* $a \in A$*, la fonction* $q \mapsto C(a,q)$ *est croissante.*

DÉFINITION 2.13 *Une* **affectation de qualité** *est une fonction* θ*, éventuellement partielle, de la forme* $\theta : A \to Q$*.*

Soit $C : A \times Q \to \mathbb{R}^+$ une fonction de durée d'exécution. Comme nous l'avons fait dans la section précédente, nous étendons la fonction C aux couples de séquences d'actions et aux affectations de qualité. Soit α une séquence d'actions de A, et θ une affectation de qualité définie sur $\mathsf{ens}(\alpha)$. Nous définissons $C(\alpha,\theta)$ de la façon suivante :

$$C(\alpha,\theta) = \sum_{1 \le i \le |\alpha|} C(\alpha(i), \theta(\alpha(i))).$$

En particulier, étant donné une action $a \in A$, la séquence $\alpha = a$ de longueur 1 permet de définir $C(a,\theta)$ comme $C(a,\theta(a))$.

DÉFINITION 2.14 *Soient* (G,C,D,Q) *un système paramétré,* α *une trace de* G *et* θ *une affectation de qualité. Nous dirons que que le couple* (α,θ) *est une* **trace** *de* (G,C,D,Q) *si* θ *est définie sur* $\mathsf{ens}(\alpha)$*. Si de plus* α *est un ordonnancement de* G*, nous dirons que* (α,θ) *est un* **ordonnancement** *de* (G,C,D,Q)*. Dans ce cas,* θ *est une fonction totale* $A \to Q$*.*

Nous dirons qu'un ordonnancement (α,θ) *est un ordonnancement* **réalisable** *de* (G,C,D,Q) *si toutes les échéances sont respectées, c'est-à-dire :*

$$D(^k\alpha) \ge C(^k\alpha,\theta).$$

Nous noterons $\mathsf{real}(G,C,D,Q)$ *l'ensemble des ordonnancements réalisables du système paramétré* (G,C,D,Q)*.*

Pour un système paramétré, un ordonnancement n'est plus seulement une séquence d'actions α, comme cela pouvait être le cas pour les systèmes simples, mais un couple (α,θ) où α est un ordonnancement du graphe G est θ une affectation de qualité définie sur A tout entier. Un ordonnancement (α,θ) est réalisable si les actions respectent leurs échéances, en considérant que la durée d'exécution de $a \in A$ est $C(a,\theta(a))$.

PROBLÈME 2.2 (ordonnancement d'un système paramétré) *Soit* (G,C,D,Q) *un système paramétré tel que* $G = (A,\prec)$*. Nous cherchons un ordonnancement* (α,θ) *de* (G,C,D,Q) *tel que :*

1. (α,θ) *est un ordonnancement réalisable de* (G,C,D,Q)*.*
2. $C(\alpha,\theta)$ *soit maximal parmi les ordonnancements réalisables de* (G,C,D,Q)*, c'est-à-dire :*

$$C(\alpha,\theta) = \mathbf{max}\{\, C(\alpha',\theta') \mid (\alpha',\theta') \in \mathsf{real}(G,C,D,Q)\}.$$

3. *L'affectation de qualité* θ *est la plus régulière possible, c'est-à-dire que les fluctuations de niveaux de qualité sont minimales parmi les ordonnancements qui maximisent* $C(\alpha,\theta)$*. Il existe différents critères pour mesurer la régularité de l'affectation de qualité* θ*. Par exemple, l'écart type ou encore la variance de l'affectation* θ *pourraient être utilisés pour mesurer la régularité de* θ*.*

Dans le problème d'ordonnancement d'un système paramétré, nous cherchons non seulement à respecter les échéances, mais aussi à utiliser au mieux les ressources de calcul. Le respect des échéances correspond simplement aux respect des contraintes temps-réel définies par l'utilisateur. En ce qui concerne l'utilisation des ressources de calcul, nous nous intéressons à deux critères.

- Le premier concerne l'utilisation de la plate-forme, que nous mesurons par la quantité $C(\alpha,\theta)$. Puisque les durées d'exécution sont des fonctions croissantes du niveau de qualité, en maximisant $C(\alpha,\theta)$ nous maximisons aussi les niveaux de qualités qui sont choisis par le contrôleur. Nous supposons de plus que la précision des traitements et calculs qui sont implémentés dans l'application est aussi une fonction croissante des niveaux de qualité.
- Le second critère est la régularité de l'affectation de qualité. En évitant les fluctuations des niveaux de qualité choisis par le contrôleur, nous cherchons à garantir une qualité constante des traitements mis en œuvre dans l'application.

Nous considérons que la combinaison des deux critères précédents permet, dans une certaine mesure, une qualité de service finale maximale.

2.2.2 Résolution à l'aide de contraintes linéaires

Nous allons donner le programme linéaire qui correspond au problème d'ordonnancement. Dans toute cette section, nous supposerons que les fonctions $q \mapsto C(a,q)$ sont linéaires. Dans le cas contraire il n'est pas possible d'exprimer le problème sous forme de programme linéaire. Nous devons également définir une fonction objectif ϕ.

Programme linéaire proposé

Le programme linéaire est très proche de celui qui a été présenté dans la section 2.1.3. Il comporte les variables libres supplémentaire q_i à valeur dans Q (voir équation 2.35), qui correspondent aux niveaux de qualité des actions a_i. Ainsi, la correspondance entre les variables q_i et l'affectation de qualité considérée est donnée par $\theta : a_i \mapsto q_i$. Les durées d'exécution sont bien entendu exprimées en fonction des niveaux de qualités q_i choisis pour les actions.

La fonction objectif $\phi : Q^n \to \mathbb{R}$ donne la qualité globale de l'ordonnancement en fonction des niveaux des qualités $q_1, \ldots, q_n$. Le choix de la fonction ϕ est traité dans la suite. Dans le programme linéaire suivant, M représente un entier assez grand, par exemple $M = \sum_{1\leq i\leq n} C(a_i, q_{max})$.

$$z_{i,j} + z_{j,i} = 1 \qquad \forall (i,j) \in \{1,\ldots,n\} \times \{1,\ldots,n\} \;.\; i \neq j \tag{2.29}$$

$$z_{i,j} = 1 \qquad \forall (i,j) \in \{1,\ldots,n\} \times \{1,\ldots,n\} \;.\; a_i \prec a_j \tag{2.30}$$

$$s_j - s_i - C(a_i,q_i) \geq M(1-z_{i,j}) \qquad \forall (i,j) \in \{1,\ldots,n\} \times \{1,\ldots,n\} \;.\; i \neq j \tag{2.31}$$

$$D(a_i) - s_i - C(a_i,q_i) \geq 0 \qquad \forall i \in \{1,\ldots,n\} \tag{2.32}$$

$$s_i \in \mathbb{R}^+ \qquad \forall i \in \{1,\ldots,n\} \tag{2.33}$$

$$z_{i,j} \in \{0,1\} \qquad \forall (i,j) \in \{1,\ldots,n\} \times \{1,\ldots,n\} \,.\, i \neq j \tag{2.34}$$
$$q_i \in Q \qquad \forall i \in \{1,\ldots,n\} \tag{2.35}$$

$$\textbf{max} \qquad \phi(q_1,\ldots,q_n). \tag{2.36}$$

Fonction objectif ϕ

Pour être conforme avec la définition du problème d'ordonnancement 2.2, nous devons choisir comme fonction objectif ϕ la fonction qui associe la somme des durées d'exécution des actions a_i à leur niveau de qualité q_i, c'est-à-dire :

$$\phi(q_1,\ldots,q_n) = \sum_{1 \leq i \leq n} C(a_i,q_i).$$

Puisque les fonctions $q_i \mapsto C(a_i,q_i)$ sont linéaires, nous obtenons une fonction objectif ϕ linéaire.

Cependant, la résolution du problème à l'aide d'un programme linéaire nous laisse la possibilité de choisir n'importe quelle fonction objectif ϕ, pourvu que celle-ci soit linéaire. Il est par exemple possible de choisir la somme pondérée de tous les niveaux de qualité, c'est-à-dire :

$$\phi(q_1,\ldots,q_n) = \sum_{1 \leq i \leq n} k_i q_i.$$

Bien entendu, dans ce cas, la solution obtenue à l'aide du programme linéaire pourra ne pas être une solution du problème d'ordonnancement 2.2. Mais la liberté de pouvoir choisir sa fonction objectif est intéressante lorsqu'on souhaite raffiner les critères d'optimisation sans changer l'algorithme de résolution du problème.

2.2.3 Complexité du problème

Le problème d'ordonnancement 2.2 peut se ramener, lorsque $|Q| = 1$, au problème d'ordonnancement 2.1. Ainsi, lorsque $|Q| = 1$, le problème est de complexité polynomiale. Nous allons voir que le problème devient NP-difficile dans le cas général, où $|Q|$ est quelconque.

PROPOSITION 2.8 *Le problème d'ordonnancement 2.2 est NP-difficile.*

Pour démontrer que le problème d'ordonnancement 2.2 est NP-difficile, nous allons réduire le problème du sac à dos, appelé également problème du BIN-PACKING, dans le problème d'ordonnancement 2.2. Le problème du sac à dos étant lui-même NP-difficile, nous pourrons conclure que le problème d'ordonnancement 2.2 est NP-difficile.

Nous donnons d'abord une définition du problème du BIN-PACKING. Une instance de ce problème est composée d'un ensemble fini d'objets O, d'une fonction $v : O \to \mathbb{N}$ donnant le volume de chaque objet, et d'un volume V qui représente le volume du sac à dos. Le but est alors de trouver un sous-ensemble d'objets $v(O')$ tel que son volume total soit borné par V et soit maximal.

DÉFINITION 2.15 (problème du BIN-PACKING) *Soit l'instance* (O,v,V) *telle que :*

1. $O = \{ o_1, \ldots, o_n \}$ *est un ensemble fini d'objets.*
2. $v : O \rightarrow \mathbb{N}$ *est une fonction associant à tout objet o son volume* $v(o)$.
3. $V \in \mathbb{N}$ *représente le volume du sac à dos.*

Etant donné une telle instance, le problème du BIN-PACKING *consiste à trouver un sous-ensemble d'objets* $O' \subseteq O$ *tel que son volume total* $v(O')$ *soit inférieur ou égal au volume du sac* V*, c'est-à-dire :*

$$V \geq v(O') = \sum_{o \in O'} v(o)$$

et tel que ce volume soit maximal, c'est-à-dire que s'il existe un sous-ensemble d'objets $O'' \subseteq O$ *de volume total* $v(O'')$ *inférieur ou égal à* V*, alors le volume total de* O' *est au moins aussi grand que celui de* O'' *:*

$$V \geq v(O'') \Rightarrow v(O') \geq v(O'').$$

Le principe de la réduction de problème du BIN-PACKING dans le problème d'ordonnancement 2.2 est d'associer à toute instance (O,v,V) du problème du BIN-PACKING une instance $(G = (A, \prec, C, D, Q) = \psi(O,v,V)$ du problème 2.2. La construction précédente peut être réalisée en temps polynomial en la taille de l'instance (O,v,V), c'est-à-dire que ψ est calculable en temps polynomial en la taille de son entrée. Il s'agit donc d'une réduction polynomiale. Cette réduction ainsi que la démonstration de la proposition 2.8 est laissée en annexe.

2.2.4 Résolution basée sur EDF

Fonction d'ordonnancement (t_s)

Nous étendons ici la notion de la fonction d'ordonnancement que nous avions donnée dans la section 2.1.2.

DÉFINITION 2.16 *Soient* (G,C,D,Q) *un système paramétré et* (α,θ) *une trace de* (G,C,D,Q)*. Nous définissons la* **fonction d'ordonnancement** *associée à* (G,C,D,Q) *de la façon suivante :*

$$t_s(\alpha,\theta)(k) = D(^k\alpha) - C(^k\alpha,\theta)$$

et :

$$t_s(\alpha,\theta) = \mathbf{min}_{1 \leq k \leq |\alpha|}\ t_s(\alpha,\theta)(k).$$

PROPOSITION 2.9 *Soit* (G,C,D,Q) *un système paramétré. Un ordonnancement* (α,θ) *de* (G,C,D,Q) *est réalisable si et seulement si* $t_s(\alpha,\theta) \geq 0$.

Preuve : La démonstration est similaire à celle donnée pour la proposition 2.1. Soit (α,θ) un ordonnancement de (G,C,D,Q). Par définition, (α,θ) est réalisable si et seulement si :

$$\begin{aligned}
& \forall k \in \{\ 1,\ldots,|\alpha|\ \}\ .\ D(^k\alpha) \geq C(^k\alpha,\theta) \\
\Leftrightarrow\ & \forall k \in \{\ 1,\ldots,|\alpha|\ \}\ .\ D(^k\alpha) - C(^k\alpha,\theta) \geq 0 \\
\Leftrightarrow\ & \forall k \in \{\ 1,\ldots,|\alpha|\ \}\ .\ t_s(\alpha,\theta)(k) \geq 0 \\
\Leftrightarrow\ & \mathbf{min}_{1\leq k\leq|\alpha|}\ t_s(\alpha,\theta)(k)\ \geq\ 0 \\
\Leftrightarrow\ & t_s(\alpha,\theta) \geq 0. \quad \square
\end{aligned}$$

PROPOSITION 2.10 *Soit (G,C,D,Q) un système paramétré. Un ordonnancement (α,θ) est un ordonnancement réalisable de (G,C,D,Q) si et seulement si α est un ordonnancement réalisable de (G,C_θ,D), où C_θ est la fonction de durée d'exécution non paramétrée suivante :*

$$C_\theta(a) = C(a,\theta(a)).$$

De plus, $t_s(\alpha,\theta) = t_s(\alpha)$, où $t_s(\alpha)$ est calculé avec la fonction d'ordonnancement associée à (G,C_θ,D).

Preuve : Nous allons montrer que $t_s(\alpha,\theta) = t_s(\alpha)$, où $t_s(\alpha)$ est calculé avec la fonction d'ordonnancement associée à (G,C_θ,D).

$$\begin{aligned}
t_s(\alpha,\theta) &= \mathbf{min}_{1\leq k\leq|\alpha|}\ D(^k\alpha) - C(^k\alpha,\theta) \\
&= \mathbf{min}_{1\leq k\leq|\alpha|}\ D(^k\alpha) - \sum_{1\leq i\leq k} C(\alpha(i),\theta(\alpha(i))) \\
&= \mathbf{min}_{1\leq k\leq|\alpha|}\ D(^k\alpha) - \sum_{1\leq i\leq k} C_\theta(\alpha(i)) \\
&= \mathbf{min}_{1\leq k\leq|\alpha|}\ D(^k\alpha) - C_\theta(^k\alpha) \\
t_s(\alpha,\theta) &= t_s(\alpha).
\end{aligned}$$

En utilisant le résultat précédent et les propositions 2.1 et 2.9, nous pouvons conclure que (α,θ) est un ordonnancement réalisable de (G,C,D,Q) si et seulement si α est un ordonnancement réalisable de (G,C_θ,D), ce qui termine la démonstration de la proposition. $\square$

Ordonnancements EDF

Nous étendons ici les résultats que nous avions présenté aux sujet des ordonnancements EDF des systèmes simples.

PROPOSITION 2.11 *Etant donnés (G,C,D,Q) un système paramétré et θ une affectation de qualité, les ordonnancement EDF de G maximisent t_s, c'est-à-dire que si α_{EDF} est un ordonnancement EDF de G par rapport à D, et α est un ordonnancement quelconque de G, alors nous avons :*

$$t_s(\alpha_{EDF},\theta) \geq t_s(\alpha,\theta).$$

Preuve : Soit θ une affectation de qualité, et C_θ la fonction de durée d'exécution non paramétrée suivante :

$$C_\theta(a) = C(a, \theta(a)).$$

Soit α_{EDF} un ordonnancement EDF de G par rapport à D, et α un ordonnancement quelconque. La proposition 2.7 nous permet d'affirmer que :

$$t_s(\alpha_{EDF}) \geq t_s(\alpha)$$

où $t_s(\alpha_{EDF})$ et $t_s(\alpha)$ sont calculés avec la fonction d'ordonnancement associée à (G, C_θ, D). D'après la proposition 2.10, nous savons que $t_s(\alpha_{EDF}, \theta) = t_s(\alpha_{EDF})$ et $t_s(\alpha, \theta) = t_s(\alpha)$. Nous concluons ainsi que :

$$t_s(\alpha_{EDF}, \theta) \geq t_s(\alpha, \theta). \quad \square$$

La proposition précédente permet de réduire la complexité exploratoire du problème d'ordonnancement. En effet, dans l'exploration des ordonnancements (α, θ) de (G, C, D, Q), nous pouvons fixer $\alpha = \alpha_{EDF}$, où α_{EDF} est un ordonnancement EDF de G par rapport à D, et effectuer l'exploration uniquement sur les affectations de qualité θ.

Algorithme d'ordonnancement

L'algorithme suivant est basé sur le résultat de la proposition 2.11. La recherche de l'ordonnancement optimal se fait en calculant tout d'abord un ordonnancement EDF de G par rapport à D, α_{EDF}. L'énumération des affectations de qualité possibles permet de trouver l'affectation maximale θ_{max} telle que l'ordonnancement $(\alpha_{EDF}, \theta_{max})$ soit réalisable, c'est-à-dire $t_s(\alpha_{EDF}, \theta_{max}) \geq 0$. Bien entendu, l'exploration des affectations de qualité θ peut être réalisée de façon intelligente, par des techniques de *branch and bound*, qui permettent de trouver l'optimum sans pour autant faire une exploration exhaustive.

```
Ordonnancer(G = (A, ≺), C, D, Q)
    α_EDF := OrdonnancerEDF(G = (A, ≺), D)
    θ_max := q_min
    pour tout θ faire
        si (t_s(α_EDF, θ) ≥ 0 ∧ θ > θ_max) faire
            θ_max := θ
        fin
    fin
    retourner (α_EDF, θ_max)
fin Ordonnancer
```

FIG. 2.10: Algorithme de recherche d'un ordonnancement optimal.

2.3 Contrôle de qualité sous incertitude

Nous allons nous intéresser dans cette section à un extension du problème d'ordonnancement de la section précédente. Dans ce nouveau problème, les durées d'exécution ne seront pas connues mais seulement bornées par des temps d'exécution pire-cas. Ainsi, la fonction C des durées d'exécution réelles ne sera pas connue, mais nous aurons connaissance d'une fonction C^{wc} telle que $C \leq C^{wc}$. De plus, les durées d'exécution réelles C pourrons être connues à posteriori, c'est-à-dire après l'exécution atomique de chaque action.

2.3.1 Définition du problème

DÉFINITION 2.17 *Nous dirons que le quadruplet $SPI(C) = (G, C^{wc}, D, Q; C)$, est un* **système paramétré incertain** *si :*

1. *La borne C^{wc} est une fonction du type $C^{wc} : A \to \mathbb{R}^+$. Celle-ci est appelée fonction de durée d'exécution* **pire-cas**.
2. *Le paramètre C est une fonction du type $C : A \times Q \to \mathbb{R}^+$. Elle fournit les durées d'exécution* **réelles** *des actions. Nous ne nous intéresserons qu'aux paramètres C tels que $C \leq C^{wc}$.*

La notion d'ordonnancement d'un système paramétré incertain est similaire à celle que nous avions donné pour les systèmes paramétrés. Un ordonnancement (α, θ) est réalisable si les échéances sont respectées par rapport aux durées d'exécution réelles, c'est-à-dire celles qui sont données par le paramètre C.

Les durées d'exécution réelles C n'étant pas connues, la recherche d'ordonnancements réalisables de $SPI(C) = (G, C^{wc}, D, Q; C)$, ne peut pas se faire en recherchant simplement les ordonnancement réalisables du système paramétré (G, C, D, Q). Puisque $C \leq C^{wc}$, les ordonnancements réalisables du système paramétré (G, C^{wc}, D, Q) sont aussi des ordonnancement réalisables du système paramétré incertain $SPI(C) = (G, C^{wc}, D, Q; C)$. Un telle façon de faire revient à approcher la fonction inconnue C par la fonction C^{wc}. Cependant, la performance d'un ordonnancement obtenu par cette méthode pourra être très éloignée de celle d'une solution optimale. En effet, en pratique les durées d'exécution pire-cas C^{wc} sont souvent fortement surestimées par rapport aux durées d'exécution réelles C.

Si les durées d'exécution réelles ne sont pas connues à priori, nous supposons par contre qu'il est possible de connaître la durée d'exécution réelle d'une action à posteriori, c'est-à-dire une fois celle-ci exécutée. Ainsi, pour résoudre le problème, nous allons mettre en œuvre des techniques de contrôle adaptatif qui consistent à prendre en compte ces informations pendant l'exécution du système. Ceci motive la définition de l'état d'un système paramétré incertain suivante.

DÉFINITION 2.18 *Etant donné un système paramétré incertain $SPI(C) = (G, C^{wc}, D, Q; C)$, nous dirons que le triplet (α, θ, t) est un* **état** *de $SPI(C)$ si :*

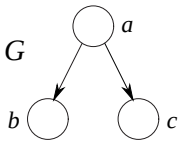
1. *α est une trace de G.*
2. *θ est une affectation de qualité définie sur $\mathsf{ens}(\alpha)$.*
3. *$t = C(\alpha, \theta)$ est la durée d'exécution totale des actions de α au niveau de qualité spécifié par θ.*

Nous dirons qu'un état (α,θ,t) *de SPI*(C) *est* **terminal** *si* α *est un ordonnancement de G.*

Lorsque les durées d'exécution étaient connues, il n'était pas nécessaire d'introduire la notion d'état intermédiaire du système, puisque nous n'avions pas à réevaluer les choix d'ordonnancement en fonction de l'évolution réelle du système. Puisque nous nous intéressons à un problème où les durées d'exécution ne sont pas connues d'avance, nous avons besoin d'introduire la notion d'état intermédiaire, appelé état du système. Les choix d'ordonnancement seront éventuellement ré-évalués sur chacun des états du système.

Pour nous, un état du système (α,θ,t) contient toutes les actions qui ont déjà été exécutées, données par la trace α, le niveau de qualité qui a été affecté à chacune de ces actions, donné par la fonction partielle θ, ainsi que la durée d'exécution totale des actions déjà exécutées, c'est-à-dire la quantité $t = C(\alpha,\theta)$. Nous supposons qu'il n'est pas possible de connaître d'avance l'intégralité de la fonction de durée d'exécution réelle C. Par contre, la durée d'exécution des actions qui ont déjà été exécutées, et donc la quantité t, est supposée connue.

graphe G



action x	$C^{wc}(x,0)$	$C^{wc}(x,1)$	$D(x)$
a	1	2	5
b	1	2	5
c	1	2	5

FIG. 2.11: Exemple de système paramétré incertain.

EXEMPLE 2.10 *Considérons le système paramétré incertain* $SPI(C) = (G, C^{wc}, D, Q, ;C)$*, tel que* $A = \{a,b,c\}$*,* $G = (A, \prec)$ *est représenté sur la figure 2.11,* $D = 5$*,* $Q = \{0,1\}$ *et pour tout* $x \in A$*,* $C^{wc}(x,q) = q+1$*. Considérons la fonction de durée d'exécution réelle* $C = C^{wc}$*. Alors* $(ab,0,2)$ *est un état de SPI*(C) *puisque ab est un trace de G, et* $C(ab,0) = 2$*.*

DÉFINITION 2.19 *Soient* $SPI(C) = (G, C^{wc}, D, Q; C)$*, un système paramétré incertain et* (α,θ,t) *un état de SPI*(C)*. Nous dirons que le couple* (α',θ') *est un* **prolongement** *à de* (α,θ,t) *dans SPI*(C) *si* α' *est un prolongement de* α *dans le graphe G et* θ' *une affectation de qualité définie sur* $\mathsf{ens}(\alpha)$*.*

Pour un état donné du système (α,θ,t), un prolongement (α',θ') fournit les choix d'ordonnancement des actions qui n'ont pas encore exécutées, c'est-à-dire l'ordre de leur exécution, donné par α', ainsi que les niveaux de qualité auxquels elles seront exécutées, donné θ'. Nous décrivons toutes les évolutions possibles du système grâce à un système de transitions étiquetées associé au système paramétré incertain.

DÉFINITION 2.20 *Soient* $SPI(C) = (G, C^{wc}, D, Q; C)$*, un système paramétré incertain.* $SPI(C)$ *induit le système de transitions étiquetées, ou* labelled transition system *(LTS), dont les états sont les états de SPI*(C)*, et la relation de transition* $\rightarrow$ *entre les états est définie de la manière suivante. Deux états* (α,θ,t) *et* (α',θ',t') *de SPI*(C) *sont en relation par le système de transition* $\rightarrow$ *si :*

1. *Il existe* $a \in A$ *tel que* $\alpha' = \alpha a$*.*

2. Il existe $q \in Q$ tel que $\theta' = \theta[a \to q]$.

Nous noterons dans ce cas $(\alpha, \theta, t) \overset{(a,q)}{\to} (\alpha', \theta', t')$.

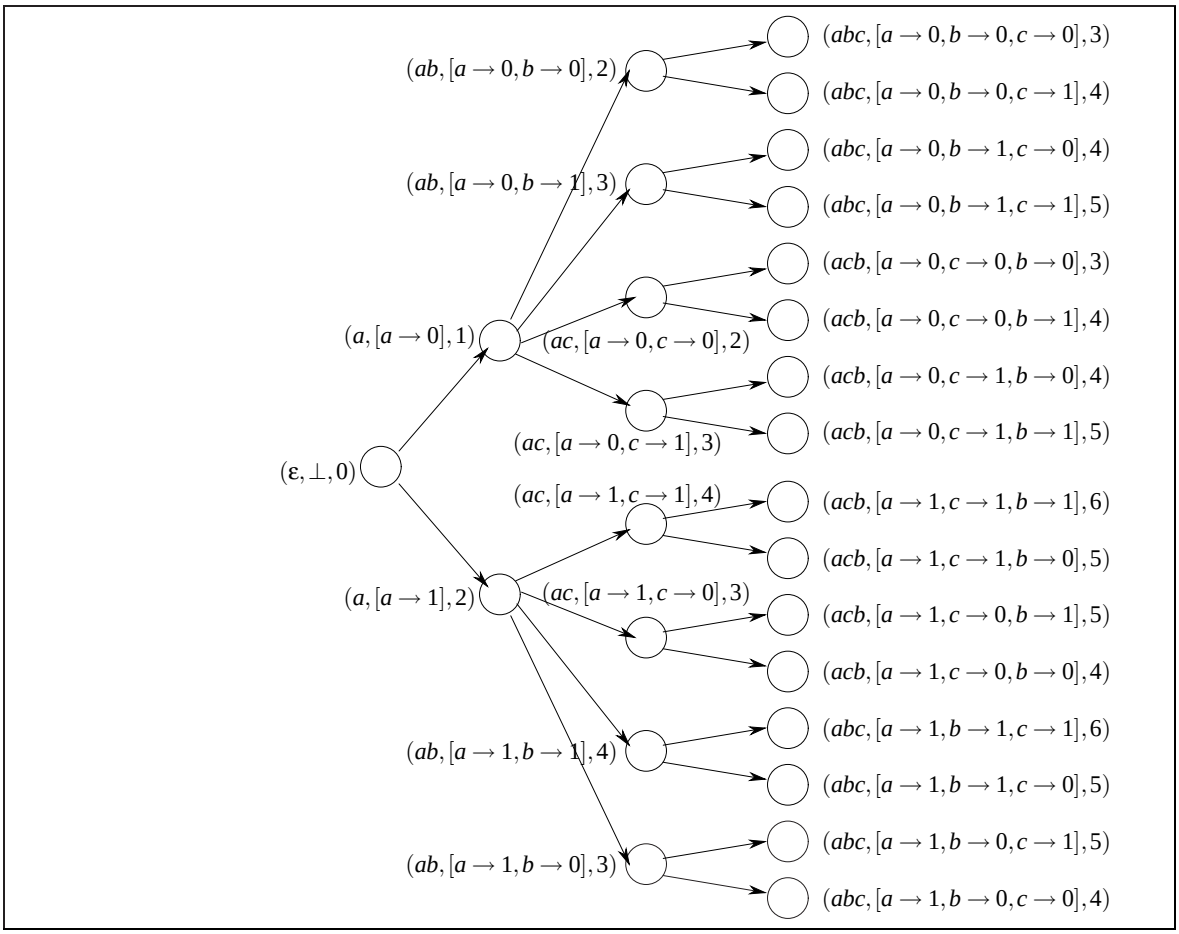


FIG. 2.12: Système de transitions étiquetées pour $C = C^{wc}$.

Le système de transitions étiquetées associé à un système paramétré incertain est un graphe orienté qui permet de décrire, étant donnée une fonction de durée d'exécution réelle C, tous les comportements possibles en fonction des choix d'ordonnancement. Remarquons que lorsque nous avons $(\alpha, \theta, t) \overset{(a,q)}{\to} (\alpha', \theta', t')$, puisque (α, θ, t) et (α', θ', t') sont des états, nous avons $t = C(\alpha, \theta)$ et $t' = C(\alpha', \theta')$, c'est-à-dire :

$$t' - t = C(a, q) \geq 0.$$

EXEMPLE 2.11 *La figure 2.12 représente le système de transitions étiquetées associé au système paramétré incertain SPI(C) de l'exemple 2.10, lorsque la fonction de durée d'exécution réelle C est telle que $C = C^{wc}$. Sur cette figure, $\perp$ désigne la fonction constante à $\perp$, c'est-à-dire définie nulle part.*

DÉFINITION 2.21 *Soient $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain, n états $(\alpha_1, \theta_1, t_1)$, …, $(\alpha_n, \theta_n, t_n)$ de $SPI(C)$, et n couples (a_1, q_1), …, (a_n, q_n) de $A \times Q$. Nous dirons que $(\alpha_1, \theta_1, t_1)$, …, $(\alpha_n, \theta_n, t_n)$ est une* **séquence d'exécution** *de $SPI(C)$ de longueur n si les états $(\alpha_1, \theta_1, t_1)$, …, $(\alpha_n, \theta_n, t_n)$ vérifient :*

1. $(\varepsilon, \perp, 0) \overset{(a_1,q_1)}{\rightarrow} (\alpha_1, \theta_1, t_1)$.
2. *Pour tout* $i \in \{ 1, \ldots, n-1 \}$ *nous avons* $(\alpha_i, \theta_i, t_i) \overset{(a_{i+1},q_{i+1})}{\rightarrow} (\alpha_{i+1}, \theta_{i+1}, t_{i+1})$.

Nous noterons une telle séquence de la façon suivante :

$$(\varepsilon, \perp, 0) \overset{(a_1,q_1)}{\rightarrow} (\alpha_1, \theta_1, t_1) \overset{(a_2,q_2)}{\rightarrow} (\alpha_2, \theta_2, t_2) \overset{(a_3,q_3)}{\rightarrow} \ldots \overset{(a_n,q_n)}{\rightarrow} (\alpha_n, \theta_n, t_n).$$

Une séquence d'exécution de longueur $|A|$ est dite **complète**.

Une séquence d'exécution correspond simplement à un chemin du système de transitions étiquetées, dont l'origine est $(\varepsilon, \perp, 0)$. Un prolongement correspond à un chemin du système de transitions dont la terminaison est un état terminal.

Pour un système paramétré incertain $SPI(C) = (G, C^{wc}, D, Q; C)$, tel que $G = (A, \prec)$, et une séquence de d'exécution $(\varepsilon, \perp, 0) \overset{(a_1,q_1)}{\rightarrow} (\alpha_1, \theta_1, t_1) \overset{(a_2,q_2)}{\rightarrow} \ldots \overset{(a_n,q_n)}{\rightarrow} (\alpha_n, \theta_n, t_n)$ de longueur $n = |A|$ correspond à l'exécution successive de a_1 à la qualité q_1, puis a_2 à la qualité q_2, …, et enfin a_n à la qualité q_n. Ainsi, une telle séquence d'exécution correspond à l'exécution du système lorsque l'on impose l'ordonnancement $(\alpha, \theta) = (a_1 a_2 \ldots a_n, \perp[a_1 \rightarrow q_1, \ldots, a_n \rightarrow q_n])$, c'est-à-dire $(\alpha, \theta) = (\alpha_n, \theta_n)$.

DÉFINITION 2.22 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain. Nous dirons que la fonction Γ est un* **contrôleur** *si Γ associe à tout état non terminal (α, θ, t) de $SPI(C)$ un couple $(a, q) = \Gamma(\alpha, \theta, t)$ tel que $(\alpha a, \theta[a \rightarrow q], t)$ soit un état de $SPI(C)$.*

Un contrôleur est associe à tout état un choix d'ordonnancement concernant la prochaine action à exécuter. Il déterminise ainsi l'exécution d'un système paramétré incertain de telle sorte que, pour une fonction de durée d'exécution C donnée, il n'existe qu'une et une seule séquence d'exécution complète possible.

DÉFINITION 2.23 *Soient $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain et Γ un contrôleur de $SPI(C)$. Nous définissons le système de transitions étiquetées associé au système contrôlé, noté $SPI(C)||\Gamma$ par :*

1. *Les états de $SPI(C)||\Gamma$ sont les mêmes que ceux de $SPI(C)$.*
2. *Deux états (α, θ, t) et (α', θ', t') sont en relation dans $SPI(C)||\Gamma$ si $(\alpha, \theta, t) \overset{(a,q)}{\rightarrow} (\alpha', \theta', t')$, c'est-à-dire que les états sont déjà en relation dans $SPI(C)$, et si $(a, q) = \Gamma(\alpha, \theta, t)$, c'est-à-dire que les choix d'ordonnancement permettant de passer de l'état (α, θ, t) à (α', θ', t') correspondent à ceux du contrôleur sur l'état (α, θ, t).*

PROBLÈME 2.3 (contrôle d'un système paramétré incertain) *Le problème d'ordonnancement auquel nous nous intéressons est le suivant. Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain. Nous cherchons un contrôleur Γ tel que pour toute fonction de durée d'exécution réelle C, l'ordonnancement (α, θ) obtenu dans le système contrôlé $SPI(C)||\Gamma$ est tel que :*

1. *(α, θ) est un ordonnancement réalisable de $SPI(C)$.*
2. *Le taux d'utilisation de la plate-forme est maximal, c'est-à-dire que la valeur $C(\alpha, \theta)$ est maximale parmi celle de tous les ordonnancements réalisables de $SPI(C)$, c'est-à-dire $C(\alpha, \theta) = \mathbf{max}_{(\alpha', \theta') \in \mathsf{real}(SPI(C))} C(\alpha', \theta')$.*
3. *L'affectation de qualité θ est régulière. Par exemple, nous chercherons à ce que l'écart type ou la variance de l'affectation θ pourraient être utilisés pour mesurer la régularité de l'affectation de qualité.*

Bien entendu, puisque les durées d'exécution ne sont pas connues d'avance, le contrôleur Γ que nous cherchons ne doit pas dépendre de C. Nous ne pourrons donc pas, avec un tel contrôleur, atteindre exactement l'optimal. Le problème est donc de trouver un contrôleur Γ dont les performances sont les plus proches possibles de l'optimal.

Nous distinguons deux approches pour construire un tel contrôleur. La première consiste à modéliser le problème de contrôle dans un formalisme donné, puis à mettre en œuvre des techniques de synthèse de contrôleur. Ces dernières permettent, à partir de la formalisation du système de d'une propriété donnée, de calculer un contrôleur qui assure que le système contrôlé vérifie la propriété. Malheureusement, la complexité prohibitive des algorithmes mis en jeu rend bien souvent le problème insoluble en pratique, dès que la taille du modèle est importante.

Nous nous orientons donc vers une autre approche, qui consiste à concevoir un contrôleur ad hoc à partir de la propriété que l'on souhaite satisfaire. Il convient alors de vérifier ensuite que le système contrôlé vérifie bien la propriété de départ. Cette approche est nettement moins coûteuse dans la mesure où ce problème de vérification est beaucoup plus simple que celui de la synthèse. D'un autre côté, concevoir un contrôleur demande une certaine dose d'intuition et de savoir-faire. De plus, dans le cas où le système contrôlé ne vérifie pas le propriété, il est nécessaire de modifier le contrôleur, et vérifier à nouveau si le système satisfait la propriété de départ avec le contrôleur modifié. Ce processus itératif peut, lui aussi, prendre un temps non négligeable.

2.3.2 Réduction du problème à la synthèse de contrôleur dans les automates temporisés

Le problème de la synthèse de contrôleur a été étudié pour différents formalismes, et en particulier celui des automates temporisés [9]. Nous proposons ici de modéliser le système paramétré incertain sous forme d'un automate temporisé particulier. Nous présentons rapidement le problème de la synthèse de contrôleur dans le cas particulier du modèle d'automate temporisé qui est proposé.

Automates temporisés et problème de la synthèse de contrôleur

Nous donnons d'abord la définition d'un automate temporisé. Nous ne donnons pas la forme générale d'un tel automate, mais le modèle minimal nécessaire à la modélisation du problème de contrôle d'un système paramétré incertain.

DÉFINITION 2.24 *L'***automate temporisé** *utilisé pour modéliser un système paramétré incertain est un n-uplet* $AT = (S, T, X, L, \mathcal{T})$ *tel que :*

Structure de contrôle.

- S *est un ensemble fini d'*états de contrôle.
- L *est un ensemble fini d'*étiquettes. *Nous supposons que ces étiquettes sont partitionnées en* $L^{\mathbf{c}}$ *et* $L^{\mathbf{u}}$.
- $T \subseteq S \times L \times S$ *est un ensemble fini de transitions étiquetées. Nous dirons que* $(s,l,s') \in T$ *est* contrôlable *si* $l \in L^{\mathbf{c}}$, *et* incontrôlable *si* $l \in L^{\mathbf{u}}$.

Temporisation.

- $X = \{ x_1, \ldots, x_n \}$ *est un ensemble fini d'*horloges.
- *La fonction* $\mathcal{T}$ *est la* fonction de temporisation *des transitions. Elle associe a toute transition* $(s,l,s') \in T$ *un couple* $\mathcal{T}(s,l,s') = (g,r) = (\mathsf{garde}(s,l,s'), \mathsf{r\text{-}z\acute{e}ro}(s,l,s'))$. *La* garde g *est un prédicat sur la valeur des horloges, c'est-à-dire une fonction boolénne du type* $(\mathbb{R}^+)^n \to \{ vrai, faux \}$. *Le sous-ensemble des horloges* $r = \mathsf{r\text{-}z\acute{e}ro}(s,l,s') \subseteq X$ *correspond aux horloges qui sont remises à zéro par la transition.*

Un automate temporisé est constitué essentiellement d'une structure de contrôle finie. Cette structure de contrôle est un automate classique, c'est-à-dire un ensemble fini d'états de de transitions étiquetées entre ces états. Nous distinguons deux types de transitions : les transitions contrôlables et incontrôlables. Les transitions contrôlables correspondent aux choix possibles du contrôleur. Dans le cas de la modélisation du problème de contrôle d'un système paramétré incertain, la temporisation est assurée par des horloges à valeur dans $\mathbb{R}^+$. Nous ajoutons également des gardes sur les transitions. Elles permettent de restreindre les valeurs possibles des horloges pour lesquelles les transitions peuvent être "tirées". Pour finir, les horloges peuvent être remise à zéro par les transitions.

DÉFINITION 2.25 (sémantique) *Soit* $AT = (S, T, X, L, \mathcal{T})$ *un automate temporisé à n horloges. L'***état** *de AT est donné par un couple* $(s, \mathbf{x}) \in S \times (\mathbb{R}^+)^n$. *L'état* $(s, \mathbf{x})$ *signifie que AT est dans l'état de contrôle s et que la valeur des horloges* x_1, …, x_n *est donnée par le vecteur* $\mathbf{x}$, *c'est-à-dire* $(x_1, \ldots, x_n) = \mathbf{x}$.

La **sémantique** *de l'automate temporisé est donnée par la relation de transition étiquetée* $\to$ *qui met les états en relation :*

$$(s,\mathbf{x}) \xrightarrow{t} (s,\mathbf{x}+t) \Leftrightarrow \forall (s,l,s') \in T \,.\, \forall t' \in]0,t[\,.\, \mathsf{garde}(s,l,s')(\mathbf{x}+t') \Rightarrow \forall t'' \in]t',t] \,.\, \mathsf{garde}(s,l,s')(\mathbf{x}+t'').$$

$$(s,\mathbf{x}) \xrightarrow{l} (s',\mathbf{x}') \Leftrightarrow \exists (s,l,s') \in T \,.\, \mathbf{x}' = \mathbf{x}\{\mathsf{r\text{-}z\acute{e}ro}(s,l,s')\} \,\wedge\, \mathsf{garde}(s,l,s')(\mathbf{x}).$$

où $\mathbf{x}\{\mathsf{r\text{-}z\acute{e}ro}(s,l,s')\} \in (\mathbb{R}^+)^n$ *désigne le vecteur* $\mathbf{x}$ *dans lequel les horloges de l'ensemble* $\mathsf{r\text{-}z\acute{e}ro}(s,l,s')$ *ont été remise à zéro.*

L'état de l'automate temporisé est définie par un état de contrôle et une affectation de la valeur des horloges. Le passage d'un état à un autre se fait soit par le passage du temps, soit en tirant une transition de l'automate. La progression du temps correspond à l'incrémentation uniforme de la valeur des horloges, sans changer l'état de contrôle de l'automate. Lorsque la garde d'une transition est vraie et devient fausse, car nous considérons que la transition devient urgente et que le temps ne peut plus progresser. Ceci nous permet de modéliser, entre autres, les durées d'exécution pire-cas d'un système paramétré incertain. Lorsqu'une transition de l'automate est tirée, l'état de contrôle est modifié en conséquence. La valeur des horloges ne change pas sauf si elles sont remises à zéro. Les transition de l'automate sont donc instantanée.

La notion d'invariant est une notion très utilisée dans description des système informatiques et leurs propriétés. Un invariant est un prédicat sur les états du système tel que, si l'état du système satisfait ce prédicat à un instant donné, alors il continue à le satisfaire par la suite. L'invariant de contrôle dans un automate temporisé est un prédicat tel que l'automate peut être amené à ne plus le satisfaire, mais seulement par une transitions contrôlables. Il est donc possible de forcer un automate temporisé à satisfaire un invariant de contrôle en agissant sur ses transitions contrôlables, c'est-à-dire en contrôlant le système.

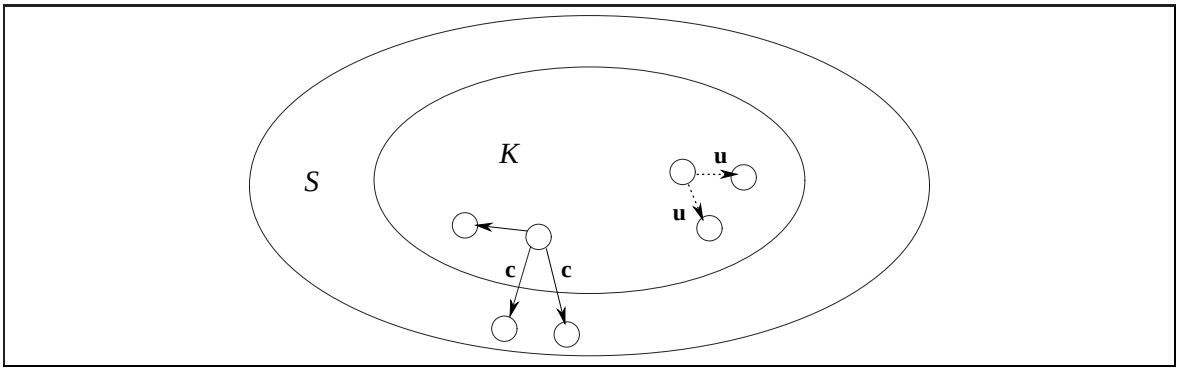


FIG. 2.13: Invariant de contrôle K.

DÉFINITION 2.26 *Soit $AT = (S,T,X,L,\mathcal{T})$ un automate temporisé et $\rightarrow$ sa sémantique. Nous dirons que le prédicat K sur les états de AT est un* **invariant de contrôle** *si :*

$$\forall (s,\mathbf{x}) \in K \,.\, (s,\mathbf{x}) \xrightarrow{l} (s',\mathbf{x}') \;\Rightarrow\; (s',\mathbf{x}') \in K \;\vee\; l \in L^{\mathbf{c}}.$$

Nous introduisons à présent la notion de prédécesseur contrôlable d'un prédicat K sur les états d'un automate temporisé. Une notion similaire est donnée dans [8] pour des modèle d'automate temporisé plus riches que celui que nous avons choisi ici.

DÉFINITION 2.27 *Soit K un prédicat sur les états d'un automate temporisé $AT = (S,T,X,L,\mathcal{T})$, et $\rightarrow$ sa sémantique. Nous appelons* **prédécesseur contrôlable** *de K le prédicat défini par :*

$$\pi(K)(s,\mathbf{x}) \Leftrightarrow \exists l \in L^{\mathbf{c}} \,.\, \exists t \in \mathbb{R}^{+} \,.\, \wedge \begin{cases} (s,x) \xrightarrow{t} (s,\mathbf{x}+t) \xrightarrow{l^{\mathbf{c}}} (s',\mathbf{x}') \;\wedge\; (s',\mathbf{x}') \in K \\ \forall t' \in [0;t] \,.\, \forall l^{\mathbf{u}} \in L^{\mathbf{u}} \,.\, (s,\mathbf{x}) \xrightarrow{t'} (s,\mathbf{x}+t') \xrightarrow{l^{\mathbf{u}}} (s',\mathbf{x}'') \;\Rightarrow\; (s',\mathbf{x}'') \in K. \end{cases}$$

La notion de prédécesseur contrôlable de contrôle permet de caractériser les invariants de contrôle. Ainsi, la proposition suivante montre qu'un invariant de contrôle est le point fixe de l'opérateur $K \mapsto K \wedge \pi(K)$.

PROPOSITION 2.12 *Soit K un prédicat sur les états d'un automate temporisé $AT = (S, T, X, L, \mathcal{T})$. Alors, K est un invariant de contrôle si et seulement si $K = K \wedge \pi(K)$.*

La démonstration de la proposition est laissée au lecteur. Pour plus de détails, il suffit de se référer à []. Le problème de la synthèse de contrôleur dans notre modèle d'automate temporisé est le suivant. Soit K le prédicat qui caractérise un propriété que nous cherchons à imposer au système en le contrôlant. Nous cherchons donc à construire un contrôleur qui maintient le système dans K, c'est-à-dire à construire un invariant de contrôle K^* tel que $K^* \Rightarrow K$. Afin de restreindre le moins possible les possibilités d'évolution du système, et notamment afin d'éviter les blocages, nous cherchons l'invariant de contrôle le plus "libéral" possible, c'est-à-dire maximal au sens de l'implication.

PROBLÈME 2.4 (synthèse de contrôleur) *. Soit $AT = (S, T, X, L, \mathcal{T})$ un automate temporisé. Soit K un prédicat sur l'état de AT. Nous cherchons un prédicat K^* tel que :*

$$\wedge \begin{cases} K^* \Rightarrow K \\ K^* = K^* \wedge \pi(K), \end{cases} \tag{2.37}$$

tel que K^* soit maximal au sens de l'implication parmi les solutions de (2.37).

Nous avons exprimé le problème de la synthèse de contrôleur dans un automate temporisé comme une équation de point fixe. Les propriétés de l'opérateur $K \mapsto K \wedge \pi(K)$ font que, si le problème a une solution, alors elle la suite de prédicat $(K_i)_{i \in \mathbb{N}}$ converge vers la solution :

$$\begin{cases} K_0 = K \\ K_{i+1} = K_i \wedge \pi(K_i), \end{cases}$$

Cette suite converge vers la solution de (2.37), ce qui permet de résoudre le problème de synthèse de contrôleur d'un automate temporisé. En pratique, les techniques de calcul symbolique évitent de travailler directement sur la sémantique infinie et assurent la convergence de la suite en un nombre finie d'étapes [8].

Automate temporisé associé à un système paramétré incertain

L'expression du problème de contrôle d'un système paramétré incertain à l'aide d'un automate temporisé nécessite à la fois la définition de l'automate temporisé qui correspond au système paramétré incertain, mais aussi de la propriété que nous cherchons à imposer au système paramétré incertain. Concernant cette dernière, nous ne nous intéressons ici qu'au respect des échéances. Nous donnerons ainsi le prédicat sur les états de l'automate temporisé qui correspond à cette propriété.

Nous ne donnons pas la formalisation complète de la construction de l'automate temporisé associé à un système paramétré incertain donné, mais nous nous contentons de donner cette construction

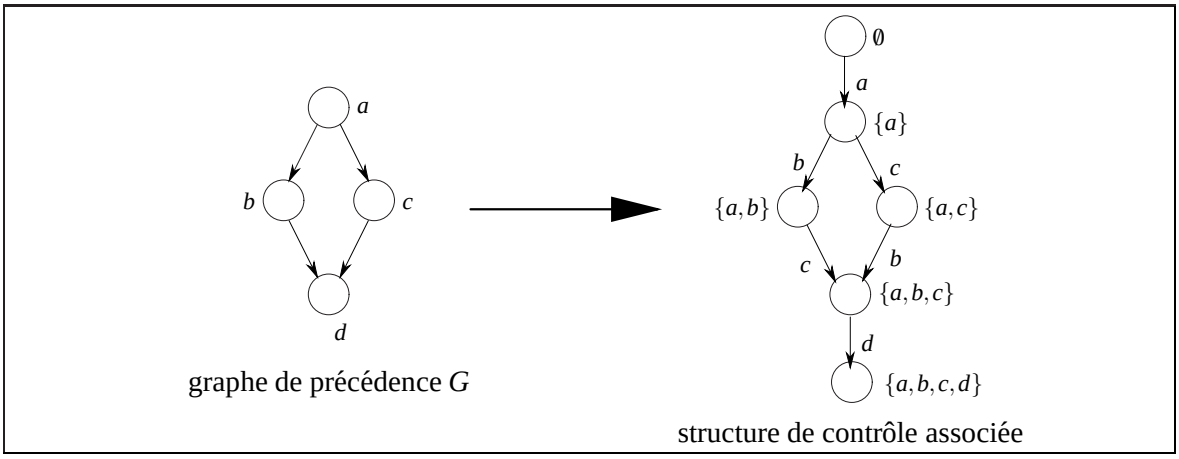


FIG. 2.14: Graphe de précédence et structure de contrôle.

pour un exemple particulier. La construction étant relativement simple, la généraliser à un système paramétré incertain quelconque sera évidente.

Considérons le système paramétré incertain $SPI(C) = (G, C^{wc}, D, Q; C)$, où le graphe de précédence $G = (A, \prec)$ est donné par la figure 2.14. Nous cherchons à construire l'automate temporisé $AT = (S, T, X, L, \mathcal{T})$ qui modélise $SPI(C)$. La première étape de cette construction concerne la structure de contrôle de l'automate. Celle-ci doit faire apparaître tous les choix de contrôle possibles, c'est-à-dire à la fois les choix d'ordonnancement mais aussi les choix des niveaux de qualité. Les premiers sont modélisés par l'ensemble des sous-ensembles d'actions fermés en arrière de G. Le choix niveau de qualité est, quant à lui, modélisé en dupliquant la structure précédent autant de fois qu'il existe de niveaux de qualité. Nous obtenons par exemple la structure de contrôle donnée dans la figure 2.15 si l'ensemble des niveaux de qualité est $Q = \{ 0, 1 \}$. Les flèches doubles de la figure 2.15 représentent deux transitions entre deux états donnés (une dans chaque sens).

Pour pouvoir temporiser l'automate, nous devons distinguer les transitions qui correspondent à des choix du contrôleur, c'est-à-dire des transition contrôlables, des transitions qui correspondent à l'exécution des actions qui ne sont pas contrôlables. Pour ce faire, il est nécessaire d'insérer un état intermédiaire et une transition après chaque choix de contrôle. La figure 2.16 illustre ce principe pour la structure de contrôle associé au niveau de qualité q. Les transitions étiquetées par un nom d'action sont les transitions contrôlables. Sur la figure, les transitions contrôlables sont représentées en traits pleins, et les transitions incontrôlables sont représentées en traits pointillés.

La temporisation proposée ici utilise deux horloges x et y. L'horloge x est utilisée pour mesurer la durée d'exécution des actions. Ainsi, x est remise à zéro avant l'exécution de chaque action, et nous assurons que la durée d'exécution d'une action est bornée par sa durée d'exécution. L'horloge y, elle, est réservée à la mesure de la durée d'exécution totale. Elle est utilisée dans le prédicat que nous cherchons à imposer à l'automate temporisé afin de vérifier si les actions respectent leurs échéances. Ainsi, l'automate final est donné dans la figure 2.15. Dans cet automate, les transitions étiquetées par des noms d'action et les transitions correspondant aux changement de niveau de qualité (arcs horizontaux sur la figure), sont les transitions contrôlables de l'automate. L'état initial de l'automate

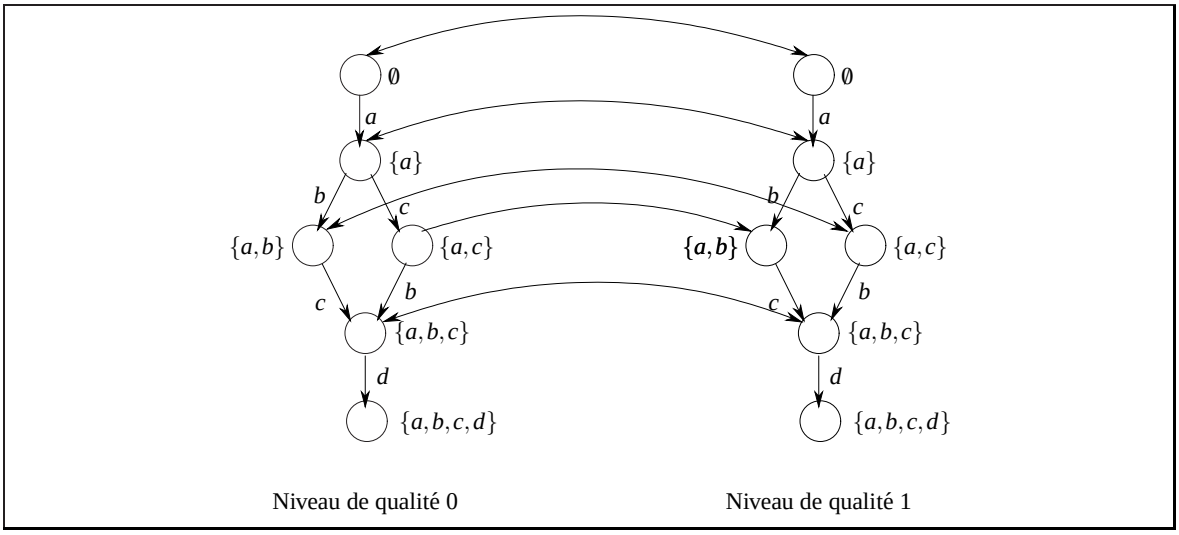


FIG. 2.15: Traduction des niveaux de qualité dans la structure de contrôle.

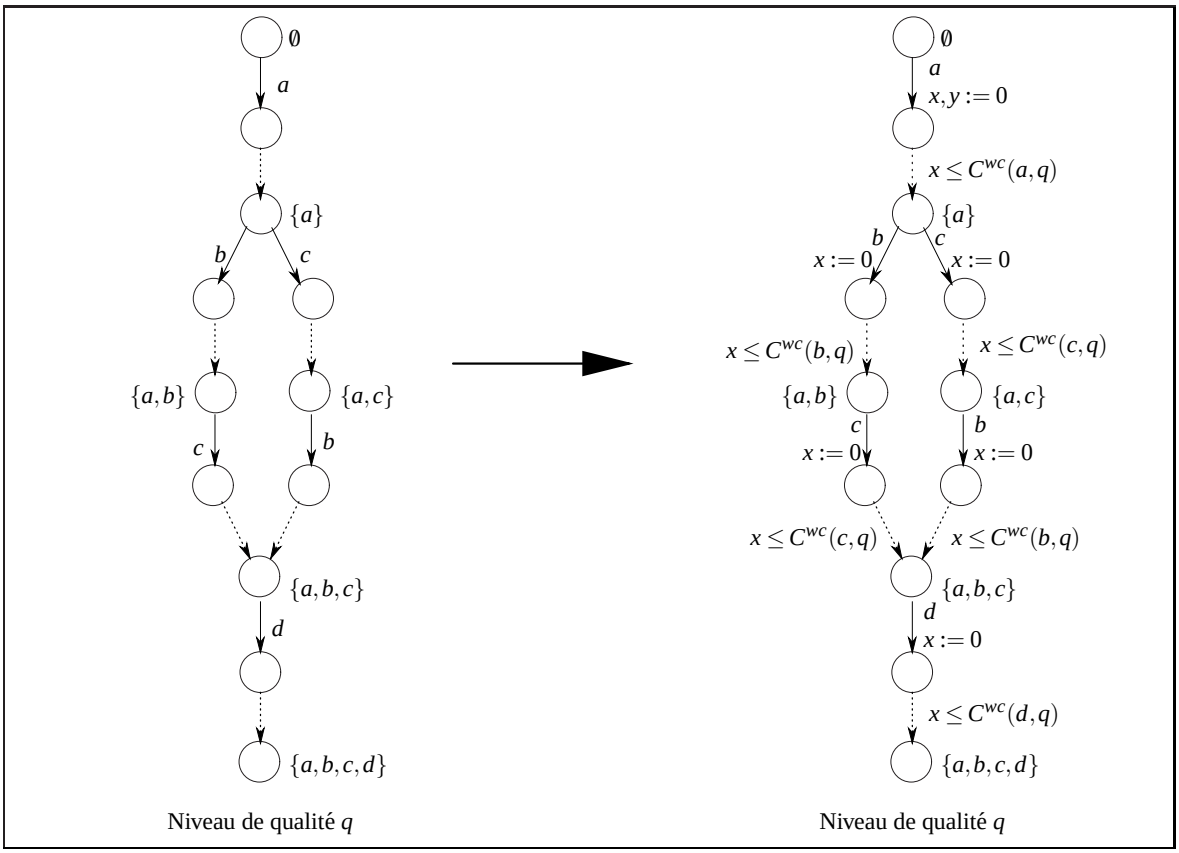


FIG. 2.16: Temporisation de la structure de contrôle.

est entouré sur la figure.

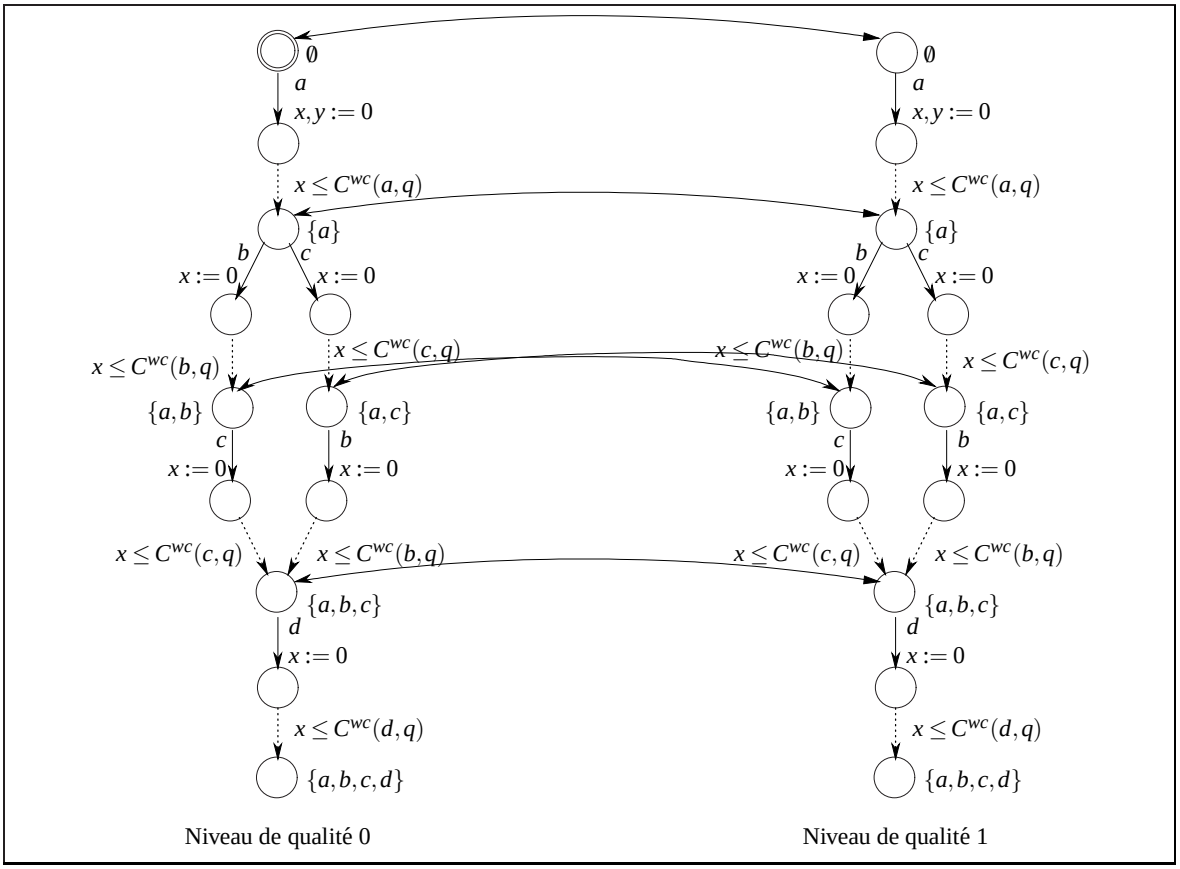


FIG. 2.17: Automate temporisé final.

Nous ne nous intéressons ici qu'au respect des échéances. Ainsi, le prédicat K sur l'état que nous cherchons à imposer à l'automate temporisé $AT = (S, T, X, L, \mathcal{T})$ est définie de la manière suivante. Pour l'état de contrôle $s \in S$ correspondant au sous-ensemble d'actions fermé en arrière F, le prédicat K est donné par :

$$K(s, \mathbf{x}) \;\Leftrightarrow\; \forall a \in F \;.\; D(a) \geq y.$$

Du fait de l'explosion combinatoire, les techniques de synthèse de contrôleur pour les automates temporisés sont souvent très lourdes, même sur des automates temporisés de taille raisonnable. Pour éviter de faire de la synthèse de contrôleur, nous préférons concevoir un contrôleur ad hoc et prouver qu'il permet au système les propriétés souhaitées comme par exemple le respect des échéances.

CHAPITRE 3

Conception d'un contrôleur de qualité de service pour la sûreté et l'optimalité

DANS le chapitre 2, nous avons défini le problème de contrôle d'un système paramétré incertain (problème 2.3). Nous avons vu, dans ce même chapitre, que les techniques de synthèse de contrôleur sont trop coûteuses pour être employées en pratique comme solution du problème de contrôle d'un système paramétré incertain. Ainsi, nous proposons ici de résoudre le problème non pas par synthèse, mais par construction d'un contrôleur ad hoc. Nous vérifions ensuite que le contrôleur proposé répond bien au problème. En particulier, nous nous intéressons à deux types de propriétés :

– des propriétés relatives à la sûreté, comme le respect des échéances ;
– des propriétés relatives à la performance, et plus particulièrement à l'exploitation optimale des ressources dans le but de maximiser la qualité de service.

A partir d'une définition générique appelée *algorithme de contrôle abstrait*, nous proposerons plusieurs instanciations du contrôleur. Ces différentes instanciations correspondent à l'utilisation de différentes politiques de contrôle. Ainsi, le reste du chapitre sera consacré à la définition et l'étude des différentes politiques de contrôle proposées.

3.1 Algorithme de contrôle abstrait

Cette première section est consacrée à la présentation et la définition du contrôleur générique Γ^X. Ce dernier est paramétré par une *politique de contrôle*. Pour un système paramétré incertain donné $SPI(C) = (G, C^{wc}, D, Q; C)$, $G = (A, \prec)$, nous donnons l'algorithme qui permet de calculer la réponse (a, q) du contrôleur Γ^X à l'état (α, θ, t) de $SPI(C)$. Cet algorithme est appelé *algorithme de contrôle abstrait*, dans la mesure où il est instancié en fonction de la politique de contrôle choisie.

3.1.1 Généralités

L'algorithme de contrôle abstrait est inspiré de l'algorithme qui a été proposé dans la section 2.2.4 pour résoudre le problème de l'ordonnancement d'un système paramétré. Il diffère cependant de ce dernier dans la mesure où il ne s'agit pas de calculer statiquement un ordonnancement complet (α, θ), mais de calculer la prochaine action à exécuter $a \in A$, et de son niveau de qualité $q \in Q$. L'algorithme de contrôle abstrait est donc bien un contrôleur au sens de la définition 2.22. Ce contrôleur est composé de deux entités qui fonctionnent en interaction : le Gestionnaire de Qualité et l'Ordonnanceur (voir figure 3.1).

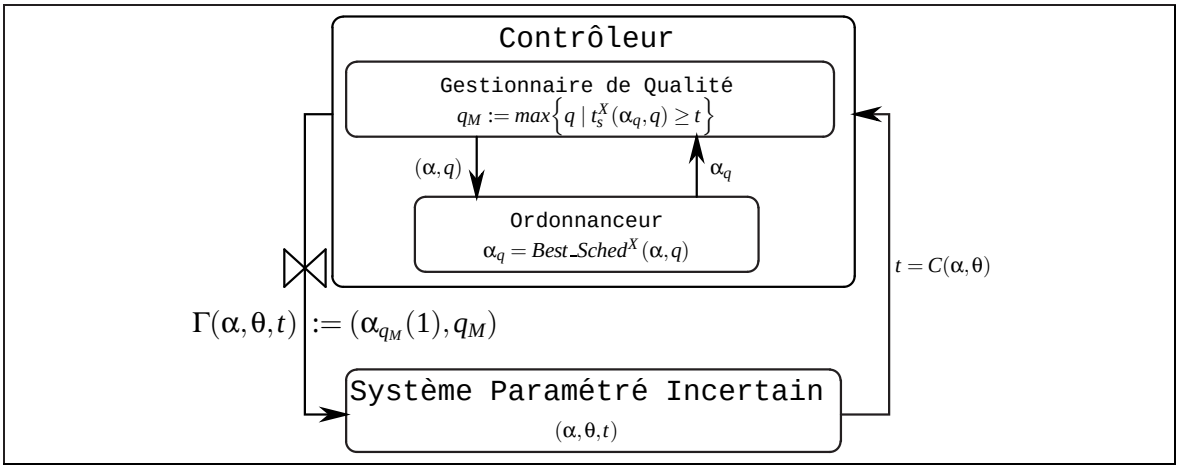


FIG. 3.1: Architecture du contrôleur.

Soit (α, θ, t) l'état courant du système paramétré incertain $SPI(C) = (G, C^{wc}, D, Q; C)$.

Gestionnaire de Qualité. Le rôle du Gestionnaire de Qualité est de déterminer le niveau de qualité courant $q \in Q$. Le choix de q est basé sur un *critère d'admission* de la forme $t_s^X(\alpha_q, q) \geq t$, dans lequel α_q représente l'ordre d'exécution des actions qui restent à exécuter, et q leur niveau de qualité. Ainsi, α_q est un prolongement de α dans G. La *fonction d'ordonnancement* t_s^X définie la *politique de gestion de qualité* du contrôleur Γ^X. Lorsque le temps théorique $t_s^X(\alpha_q, q)$ dépasse le temps-réel t, nous considérons que α_q est admissible au niveau de qualité q, c'est-à-dire que l'application est suffisamment en avance pour pouvoir ordonnancer au niveau de qualité q et selon l'ordre imposé par α_q.

Ordonnanceur. Le rôle de l'Ordonnanceur est de calculer, pour chaque niveau de qualité $q \in Q$, un prolongement optimisé $\alpha_q = \mathit{Best\_Sched}^X(\alpha, q)$ pour le niveau de qualité q. La fonction $\mathit{Best\_Sched}^X$ définit la *politique d'ordonnancement* du contrôleur Γ^X. Puisque le critère d'admission du Gestionnaire de Qualité est de la forme $t_s^X(\alpha_q, q) \geq t$, l'optimisation consiste à trouver un prolongement α_q qui maximise $t_s^X(\alpha_q, q)$. Ceci permet de maximiser l'ensemble des valeurs du temps-réel t pour lesquelles le α_q est admissible au niveau de qualité q.

Bien que nous limitions l'exploration aux prolongements de la forme (α_q, q), c'est-à-dire dont le niveau de qualité est constant, nous verrons dans le chapitre 5 que obtenons quand même de bons résultats quant à l'utilisation des ressources de calcul. En effet, le contrôle étant réalisé à grain fin, les choix du contrôleur, et en particulier ceux qui concernent le niveaux de qualité, sont remis en cause et réévalués après l'exécution de chaque action, en fonction de l'état courant du système paramétré incertain.

```
Γ^X(α, θ, t)
    q_m := q_min
    pour tout q ∈ Q faire
        α_q := Best_Sched^X(α, q)
        si (t_s^X(α_q, q) ≥ t ∧ q > q_m) faire
            q_m := q
        fin
    fin
    retourner (a, q) = (α_{q_m(1)}, q_m)
fin
```

FIG. 3.2: Algorithme de calcul du contrôleur Γ^X.

L'algorithme de contrôle abstrait est donné par la figure 3.2. L'algorithme prend en entrée l'état (α, θ, t) du système paramétré incertain considéré $SPI(C) = (G, C^{wc}, D, Q; C)$. Pour chaque niveau de qualité q, l'Ordonnanceur calcule d'abord les prolongements $(\alpha_q, q) = (\mathit{Best\_Sched}^X(\alpha, q), q)$ à l'état (α, θ, t). Le Gestionnaire de Qualité détermine ensuite quels sont les prolongements admissibles par rapport à la politique de gestion de qualité. Pour ce faire, il évalue le critère d'admission $t_s^X(\alpha_q, q) \geq t$. L'algorithme de contrôle abstrait renvoie finalement la première action $a = \alpha_{q_m}(1)$ et le niveau de qualité q_m du prolongement (α_{q_m}, q_m), où q_m est le niveau de qualité maximal parmi les niveaux de qualité q pour lesquels le prolongement (α_q, q) est admissible. S'il n'existe pas de prolongement admissible, c'est la qualité minimale q_{min} qui est choisie. Dans la suite, nous définissons formellement la notion de politique de contrôle. Nous proposons ensuite plusieurs instanciations du contrôleur Γ^X, qui sont Γ^{sf}, Γ^{sp} et Γ^{mx}, correspondant respectivement aux politiques de contrôle *sûre*, *simple*, et *mixte*. Le chapitre 4 abordera le problème de l'efficacité de l'implémentation des contrôleurs Γ^{sp} et Γ^{mx}.

3.1.2 Politique de contrôle

Dans la section 3.1.1, nous avons donné l'algorithme de contrôle abstrait qui permet de calculer le contrôleur Γ^X. L'algorithme proposé est générique dans la mesure où il paramétré par une politique de contrôle. Nous proposons ici de définir précisément la notion de politique de contrôle. Le contrôleur étant composé d'un Gestionnaire de Qualité et d'un Ordonnanceur, une politique de contrôle se décompose en deux parties : une politique de gestion de qualité et une politique d'ordonnancement.

Politique de gestion de qualité

DÉFINITION 3.1 *Une* **politique de gestion de qualité** *est entièrement définie par une fonction d'ordonnancement t_s^X de la forme $t_s^X : A^* \times \Theta \to \mathbb{R}^+$, telle que $t_s^X(\alpha,\theta)$ est définie pour toute séquence d'actions α et toute affectation de qualité θ définie sur $\mathsf{ens}(\alpha)$.*

Soient (α,θ) un prolongement d'un état de $SPI(C)$, a et b deux actions telles que $\alpha(i) = a$ et $\alpha(j) = b$. Nous dirons que l'échéance $D(a)$ de a est plus **critique** *que l'échéance $D(b)$ de b dans le prolongement (α,θ) par rapport à la fonction d'ordonnancement t_s^X si :*

$$t_s^X(\alpha,\theta)(i) < t_s^X(\alpha,\theta)(j).$$

De plus, nous appellerons **échéance critique** *de (α,θ) par rapport à la fonction d'ordonnancement t_s^X l'échéance $D(\alpha(j))$ de l'action $\alpha(j)$ telle que $j = \mathbf{max}\{\ k \mid t_s^X(\alpha,\theta)(k) = t_s^X(\alpha,\theta)\}$.*

Une politique de gestion de qualité est caractérisée par le choix d'une fonction d'ordonnancement t_s^X, sur laquelle est basé le critère d'admission $t_s^X(\alpha_q, q) \geq t$ du Gestionnaire de Qualité. En pratique, la fonction t_s^X est construite à partir d'une fonction de durée d'exécution C^X, de la même façon que nous avions définie la fonction t_s à partir de la fonction C dans la section 2.2.4. Soit C^X une fonction partielle du type $A^* \times \Theta \to \mathbb{R}^+$, telle que $C^X(\alpha,\theta)$ soit définie pour toute séquence d'actions α et toute affectation de qualité θ définie sur $\mathsf{ens}(\alpha)$. Dans ce cas, nous définissons la fonction d'ordonnancement t_s^X associée à C^X comme suit :

$$t_s^X(\alpha,\theta)(k) = D(^k\alpha) - C^X(^k\alpha,\theta)$$

et

$$t_s^X(\alpha,\theta) = \mathbf{min}_{1\leq k\leq|\alpha|}\ t_s^X(\alpha,\theta)(k).$$

Dans la section 2.2.1, nous avons vu comment étendre une fonction de durée d'exécution quelconque du type $A \times Q \to \mathbb{R}^+$ en une fonction partielle du type $A^* \times \Theta \to \mathbb{R}^+$, définie sur les couples (α,θ) tels que α soit une séquence d'actions et θ une affectation de qualité définie sur $\mathsf{ens}(\alpha)$. Remarquons qu'ici, la fonction $C^X : A^* \times \Theta \to \mathbb{R}^+$ qui sert à bâtir la fonction d'ordonnancement t_s^X n'est pas forcément obtenue par extension d'une fonction $C^X : A \times Q \to \mathbb{R}^+$. En particulier, nous verrons que les fonctions C^X considérées ne vérifient pas forcément $C^X(\alpha_1\alpha_2,\theta) = C^X(\alpha_1,\theta) + C^X(\alpha_2,\theta)$.

DÉFINITION 3.2 *Soit $C^X : A^* \times \Theta \to \mathbb{R}^+$ une fonction de durée d'exécution. Nous dirons que C^X est* **croissante** *si pour toutes séquences α_1 α_2 et toute affectation de qualité définie sur $\mathsf{ens}(\alpha_1\alpha_2)$ nous avons :*

$$C^X(\alpha_1, \theta) \leq C^X(\alpha_1\alpha_2, \theta).$$

Une fonction de durée d'exécution croissante C^X est telle que la durée d'exécution d'une séquence augmente si des actions lui sont ajoutées, ce qui semble une hypothèse raisonnable. Dans la suite, les fonctions C^X qui serviront à bâtir les fonctions d'ordonnancement proposées seront toujours croissantes.

PROPOSITION 3.1 *Soit C^X une fonction de durée d'exécution croissante, et t_s^X la fonction d'ordonnancement associée à C^X. Alors, nous avons :*

$$t_s^X(\alpha, \theta) = \mathbf{min}_{k \in \mathsf{critique}_{D(\alpha)}}\, t_s^X(\alpha, \theta)(k).$$

où $\mathsf{critique}_D(\alpha)$ est le même sous-ensemble que celui qui est donné dans la définition 2.9, c'est-à-dire $\mathsf{critique}_D(\alpha) = \{\ k \mid \forall k' > k\ .\ D(^{k'}\alpha) > D(^{k}\alpha)\ \}$.

Nous ne détaillons pas la preuve de la proposition précédente car elle est similaire à celle que nous avons donnée pour la proposition 2.2. Cette proposition permet de simplifier le calcul de la fonction t_s^X lorsque cette dernière est basée sur une fonction de durée d'exécution C^X croissante. Cette simplification est faite en éliminant les échéances dont nous sommes certains qu'elles ne sont pas critiques pour la séquence considérée.

La fonction d'ordonnancement t_s^X nous permet d'introduire la notion de prolongement *admissible* par rapport à une politique de gestion de qualité et un état du système. Cette notion est proche de la notion d'ordonnancement réalisable présentée dans la section 2.3.1. Elle diffère cependant de cette dernière sur les deux points suivants :

- la notion de prolongement admissible est une notion dynamique, c'est-à-dire par rapport à un état donné du système ;
- pour déterminer si un prolongement est admissible, nous utilisons la fonction d'ordonnancement t_s^X associée à la politique de gestion de qualité, et non la fonction d'ordonnancement t_s qui ne peut pas être calculée.

DÉFINITION 3.3 *Soient $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain, et (α, θ, t) un état de $SPI(C)$. Nous dirons que le prolongement (α_q, q) de (α, θ, t) dans $SPI(C)$ est* **admissible** *par rapport à une politique de gestion de qualité si $t_s^X(\alpha_q, q) \geq t$.*

Pour un état donné du système (α, θ, t), le contrôleur Γ^X choisit d'exécuter la première action $\alpha_{q_m}(1)$ au niveau de qualité q_m, où (α_{q_m}, q_m) est le prolongement qui maximise q parmi les prolongements (α_q, q) admissibles. Ainsi, la politique de gestion de qualité est construite de telle sorte que, si un prolongement (α_q, q) est admissible à l'état (α, θ, t), alors :

- exécuter la première action $\alpha(1)$ à la qualité q ne remet pas en cause le respect des échéances ;

– le prolongement (α_q, q) a de "forte chance" de pouvoir être exécuté tout en respectant les échéances, afin d'assurer la régularité des niveaux de qualité ; bien entendu, suivant les durées d'exécution réelles, il ne sera pas forcément possible de terminer l'exécution en ordonnançant selon (α_q, q), et en particulier au niveau de qualité q.

Nous verrons dans la section 3.2 que le premier point correspond à une propriété de sûreté de la politique de gestion de qualité. Concernant le second point, nous introduisons la notion de politique de gestion de qualité *conservatrice* par rapport à des estimations des durées d'exécution.

DÉFINITION 3.4 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain. Nous dirons qu'une politique de gestion de qualité est* **conservatrice** *par rapport à une fonction de durée d'exécution $C^Y : A \times Q \to \mathbb{R}^+$ si pour toute séquences d'actions α_1 et α_2, et pour tout niveau de qualité $q \in Q$, nous avons :*

$$t_s^X(\alpha_1\alpha_2, q) \geq t \ \wedge\ C(\alpha_1, q) = C^Y(\alpha_1, q) \ \Rightarrow\ t_s^X(\alpha_2, q) \geq t + C(\alpha_1, q).$$

Politique d'ordonnancement

Dans ce qui précède, nous avons donné la définition d'une politique de gestion de qualité. Afin de définir complètement une politique de contrôle, nous devons à présent introduire la notion de politique d'ordonnancement.

DÉFINITION 3.5 *Soient $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain. Une* **politique d'ordonnancement** *est définie par une fonction partielle $Best\_Sched^X$ de type $A^* \times Q \to A^*$, telle que pour toute trace α de G et tout niveau de qualité $q \in Q$, la séquence $\alpha_q = Best\_Sched^X(\alpha, q)$ est prolongement de α dans le graphe G.*

Nous dirons que la politique d'ordonnancement est une politique d'ordonnancement **EDF** *pour le niveau de qualité q si pour toute trace α de G, le prolongement $\alpha_q = Best\_Sched^X(\alpha, q)$ est un prolongement EDF. Nous dirons que la politique d'ordonnancement est une politique d'ordonnancement* **EDF** *si elle est EDF par rapport à tous les niveaux de qualité $q \in Q$.*

Nous dirons que la politique d'ordonnancement est **optimale** *par rapport au niveau de qualité q, si pour toute trace α de G le prolongement $\alpha_q = Best\_Sched^X(\alpha, q)$ de α dans G maximise la fonction d'ordonnancement t_s^X, c'est-à-dire que pour tout prolongement α'_q de α dans G, nous avons $t_s^X(\alpha_q, q) \geq t_s^X(\alpha'_q, q)$. Nous dirons que la politique d'ordonnancement est* **optimale** *si elle est optimale par rapport à tous les niveaux de qualité $q \in Q$.*

Une politique d'ordonnancement est définie par une fonction $Best\_Sched^X : A^* \times Q \to A^*$. Cette dernière prend en entrée une trace α du graphe G, correspondant aux actions qui ont déjà été exécutées, et un niveau de qualité $q \in Q$. En retour, la fonction $Best\_Sched^X$ fournit un prolongement $\alpha_q = Best\_Sched(\alpha, q)$ de la trace α dans G, c'est-à-dire un ordonnancement des actions qui n'ont pas encore été exécutées. Lorsque la politique d'ordonnancement est optimale, chaque prolongement α_q est "optimisé" par rapport au niveau de qualité q. En effet, en maximisant $t_s^X(\alpha_q, q)$, une

politique d'ordonnancement optimale permet de maximiser l'ensemble des niveaux de qualité admissibles. Ceci conduit le gestionnaire de qualité à choisir des niveaux de qualité plus élevés. Il s'agit donc d'une optimisation locale, dans le sens où nous cherchons un ordonnanceur qui conduit à activer un niveaux de qualité élevé à un instant donné. La politique de gestion de qualité doit donc être conçue de façon à ce que cette optimisation locale conduise à une optimisation globale.

Pour certaines politiques de contrôle, le calcul ou l'implémentation d'une politique d'ordonnancement optimale peut être se révéler trop coûteux ou trop complexe en pratique. Dans ce cas, nous chercherons des approximations de la politique d'ordonnancement optimale, en nous assurant que la politique d'ordonnancement approchante vérifie certaines propriétés essentielles, comme la sûreté dont il est question dans la section 3.2. En particulier, nous nous intéresserons à l'approximation qui consiste à utiliser une politique d'ordonnancement *statique*.

DÉFINITION 3.6 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain. Nous dirons que la politique d'ordonnancement est* **statique** *s'il existe un ordonnancement α_0 du graphe G tel que pour tout préfixe $\alpha =^k \alpha_0$ de G nous avons :*

$$Best\_Sched^X(\alpha, q) = {\alpha_0}^{k+1}.$$

Une politique d'ordonnancement statique est très intéressante pour deux raisons. D'une part l'implémentation d'un contrôleur basé sur une politique d'ordonnancement statique peut être réalisée de manière plus efficace, puisqu'il est possible, en pré-calculant une quantité raisonnable d'informations, d'accélérer de façon importante l'exécution de l'algorithme de contrôle abstrait. D'autre part, il est plus facile de prévoir le comportement et donc les performances d'un contrôleur basé sur une politique d'ordonnancement statique. Bien entendu, l'utiliserons d'une politique d'ordonnancement statique peut dégrader les performances générales du contrôleur, notamment en terme d'utilisation des ressources de calcul. Ainsi nous utiliserons une telle politique d'ordonnancement uniquement lorsqu'il sera possible de maintenir un niveau de performance acceptable.

DÉFINITION 3.7 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain, et Nous dirons qu'un niveau de qualité q est admissible à l'état (α, θ, t) par rapport à une politique de gestion de qualité et une politique d'ordonnancement données si la fonction d'ordonnancement t_s^X et $t_s^X(\alpha_q, q) \geq t$, où $\alpha_q = Best\_Sched^X(\alpha, q)$.*

3.2 Politique de contrôle sûre

Nous allons voir ici comment contraindre la politique de gestion de qualité et l'Ordonnanceur $Best\_Sched^X$ afin d'assurer que le système paramétré incertain contrôle respecte ses échéances. Nous définissons d'abord la politique de gestion de qualité sûre.

3.2.1 Politique de gestion de qualité sûre

Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain. Comme nous l'avons fait dans le chapitre 2, nous étendons la fonction du durée d'exécution pire-cas $C^{wc} : A \times Q \rightarrow \mathbb{R}^+$ aux couples (α, θ) tels que α soit une séquence d'actions et θ soit une affectation de qualité définie sur $\mathsf{ens}(\alpha)$, de la manière suivante :

$$C^{wc}(\alpha, \theta) = \sum_{1 \leq i \leq |\alpha|} C^{wc}(\alpha(i), \theta(\alpha(i))).$$

Nous obtenons ainsi une fonction de durée d'exécution pire-cas C^{wc} du type $A^* \times \Theta \rightarrow \mathbb{R}^+$.

DÉFINITION 3.8 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain. Nous appelons fonction de durée d'exécution* **sûre** *la fonction partielle $C^{sf} : A^* \times \Theta \rightarrow \mathbb{R}^+$ définie pour toute séquence d'actions α et toute affectation de qualité θ définie sur $\mathsf{ens}(\alpha)$, par :*

$$C^{sf}(\alpha, \theta) = C^{wc}({}^1\alpha, \theta) + C^{wc}(\alpha^2, q_{min}).$$

Le calcul de la fonction de durée d'exécution sûre C^{sf} est basé sur les durées d'exécution pire-cas C^{wc} afin de prendre en compte le pire scénario possible. Pour un prolongement donné (α, θ), le calcul de $C^{sf}(\alpha, \theta)$ est réalisé en considérant le niveau de qualité $\theta(\alpha(1))$ pour la première action $\alpha(1)$, puis le niveau de qualité minimal q_{min} pour les autres actions, c'est-à-dire $\alpha(2), \ldots, \alpha(|\alpha|)$. En effet, même dans l'hypothèse du pire scénario $C = C^{wc}$, le contrôleur peut toujours réduire le niveau de qualité après l'exécution de la première action $\alpha(1)$, et au pire la fixer au niveau minimal q_{min}, pour assurer le respect des échéances. Comme toutes les fonctions de durée d'exécution que nous allons voir, la fonction C^{sf} est une fonction de durée d'exécution croissante.

PROPOSITION 3.2 *La fonction de durée d'exécution sûre $C^{sf} : A^* \times \Theta \rightarrow \mathbb{R}^+$ est croissante au sens de la définition 3.2.*

Preuve de la proposition : Soient α_1 et α_2 deux séquences d'actions, et θ une affectation de qualité définie sur $\mathsf{ens}(\alpha_1\alpha_2)$. Alors :

$$\begin{aligned} C^{sf}(\alpha_1\alpha_2, \theta) &= C^{wc}(\alpha_1(1), \theta) + C^{wc}((\alpha_1\alpha_2)^2, q_{min}) = C^{wc}(\alpha_1(1), \theta) + C^{wc}({\alpha_1}^2, q_{min}) + C^{wc}(\alpha_2, q_{min}) \\ &= C^{sf}(\alpha_1, \theta) + C^{wc}(\alpha_2, q_{min}) \geq C^{sf}(\alpha_1, \theta). \quad \square \end{aligned}$$

DÉFINITION 3.9 *La* **politique de gestion de qualité sûre** *est donnée par la fonction partielle t_s^{sf} : $A^* \times \Theta :\rightarrow \mathbb{R}$ associée à la fonction de durée d'exécution sûre C^{sf}.*

La politique de gestion de qualité sûre sert de base à la construction de contrôleurs sûrs, c'est-à-dire qui assurent à coup sûr le respect des échéances. Soient $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain, (α, θ, t) un état de $SPI(C)$ et (α_q, q) un prolongement à l'état (α, θ, t). La fonction d'ordonnancement t_s^{sf} associée à C^{sf} est telle que la condition $t_s^{sf}(\alpha_q, q) \geq t$ garantie :

– que l'action $a = \alpha_q(1)$ peut être exécutée au niveau de qualité q, tout en respectant à coup sûr son échéance $D(a)$;
– et qu'une fois a exécutée au niveau de qualité q, il existera un prolongement (par exemple $({\alpha_q}^2, q_{min})$) tel que les échéances $D(\alpha_q(2)), \ldots, D(\alpha_q(|\alpha_q|))$ des actions qui restent à exécuter soient respectées.

Cette propriété essentielle de la politique de gestion de qualité sûre est développée plus en détail dans la proposition 3.4. Cette dernière donne les conditions qui permettent d'être certain que le système paramétré incertain contrôlé respecte ses échéances. Avant de présenter et expliquer cette proposition, nous donnons le résultat suivant à propos de la politique de gestion de qualité sûre.

PROPOSITION 3.3 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain, (α, θ, t) un état de $SPI(C)$ et (α_q, q) un prolongement à l'état (α, θ, t). Alors, nous avons :*

$$t_s^{sf}(\alpha_q, q) \geq t \;\Rightarrow\; t_s^{sf}({\alpha_q}^2, q_{min}) \geq t + C^{wc}(\alpha_q(1), q).$$

Preuve de la proposition : Nous avons :

$$\begin{array}{rcl} t_s^{sf}(\alpha_q, q) \geq t & \Rightarrow & \forall k \in \{\, 1, \ldots, |\alpha_q| \,\} \,.\, t_s^{sf}(\alpha_q, q)(k) \geq t \\ & \Rightarrow & \forall k \in \{\, 1, \ldots, |\alpha_q| \,\} \,.\, D(\alpha_q(k)) - C^{sf}(\alpha_q[1,k], q) \geq t. \end{array}$$

En utilisant la définition de C^{sf}, nous obtenons :

$$\begin{array}{rcl} t_s^{sf}(\alpha_q, q) \geq t & \Rightarrow & \forall k \in \{\, 1, \ldots, |\alpha_q| \,\} \,.\, D(\alpha_q(k)) - C^{wc}(\alpha_q(1), q) - C^{sf}(\alpha_q[2,k], q_{min}) \geq t \\ & \Rightarrow & \forall k \in \{\, 2, \ldots, |\alpha_q| \,\} \,.\, D(\alpha_q(k)) - C^{sf}(\alpha_q[2,k], q_{min}) \geq t + C^{wc}(\alpha_q(1), q) \\ & \Rightarrow & \forall k \in \{\, 1, \ldots, |{\alpha_q}^2| \,\} \,.\, D(({\alpha_q}^2)(k)) - C^{sf}(({\alpha_q}^2)[1,k], q_{min}) \geq t + C^{wc}(\alpha_q(1), q) \\ & \Rightarrow & t_s^{sf}({\alpha_q}^2, q_{min}) \geq t + C^{wc}(\alpha_q(1), q). \quad \square \end{array}$$

Soit (α, θ, t) un état du système paramétré incertain $SPI(C) = (G, C^{wc}, D, Q; C)$, et (α_q, q) un prolongement à l'état (α, θ, t). Soit $(\alpha a, \theta[a \to q], t + C(a, q))$ un état consécutif à (α, θ, t), après avoir exécuté l'action $a = \alpha_q(1)$ au niveau de qualité q. Le prolongement (α_q, q) est admissible par rapport à la politique de gestion de qualité sûre s'il est possible d'exécuter la prochaine action a au niveau de qualité q, tout en assurant que celle-ci respecte son échéance $D(a)$, et qu'il existera un prolongement admissible après son exécution. En effet, d'après la proposition 3.3, si le prolongement (α_q, q) est admissible nous avons :

$$t_s^{sf}({\alpha_q}^2, q_{min}) \geq t + C^{wc}(\alpha_q(1), q) \geq t + C(a, q).$$

Autrement dit le prolongement $({\alpha_q}^2, q_{min})$ est admissible par rapport à la politique d'ordonnancement sûre. Pour autant, cette condition d'admissibilité n'assure pas de que l'Ordonnanceur trouve un tel prolongement après l'exécution de a au niveau de qualité q. C'est ce qui peut se produire si la politique d'ordonnancement utilisée n'est pas optimale. Au contraire, avec une politique d'ordonnancement optimale par rapport au niveau de qualité q_{min}, le fait que le prolongement (α^2, q_{min}) soit admissible

nous permet de conclure que l'Ordonnanceur calcule aussi un prolongement admissible pour le niveau de qualité minimal, puisque nous avons :

$$t_s^{sf}(Best\_Sched^{sf}(\alpha a, q_{min}), q_{min}) \geq t_s^{sf}(\alpha_q{}^2, q_{min}) \geq t + C^{wc}(a,q),$$

où $Best\_Sched^{sf}$ est une politique d'ordonnancement optimale par rapport au niveau de qualité minimal q_{min}.

3.2.2 Politiques d'ordonnancement sûres

Nous avons vu que la politique de gestion de qualité sûre utilisée conjointement avec une politique d'ordonnancement optimale assure que, si un niveau de qualité est admissible à un instant donné, alors il existera toujours un niveau de qualité admissible.

PROPOSITION 3.4 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain. Soit Γ^{sf} le contrôleur basé sur la politique de gestion de qualité sûre et une politique d'ordonnancement optimale par rapport à la politique de gestion de qualité sûre. S'il existe un ordonnancement α de G tel que (α, q_{min}) soit un ordonnancement réalisable du système paramétré $SPI(C^{wc})$, alors le contrôleur Γ^{sf} calcule des ordonnancements réalisables du système paramétré incertain $SPI(C)$.*

La démonstration de la proposition précédente est faite dans le cas plus général de la proposition 3.6. En particulier, nous verrons que la politique d'ordonnancement utilisée n'a pas besoin d'être optimale par rapport à tous les niveaux de qualité mais simplement par rapport au niveau de qualité minimal q_{min}. Le résultat suivant permet de calculer facilement des ordonnancement optimaux par rapport au niveau de qualité minimal : il s'agit des ordonnancements EDF.

PROPOSITION 3.5 *Une politique d'ordonnancement EDF est optimale par rapport au niveau de qualité minimal et la politique de gestion de qualité sûre.*

Preuve : La démonstration est similaire à celle qui a été donnée pour la proposition 2.11. Soit $C^{wc}_{q_{min}} : A \rightarrow \mathbb{R}^+$ la fonction définie par :

$$C^{wc}_{q_{min}}(a) = C^{wc}(a, q_{min}).$$

Soit α_{EDF} un ordonnancement EDF de G par rapport à D, et α un ordonnancement quelconque. La proposition 2.7 nous permet d'affirmer que :

$$t_s(\alpha_{EDF}) \geq t_s(\alpha)$$

où $t_s(\alpha_{EDF})$ et $t_s(\alpha)$ sont calculés avec la fonction d'ordonnancement associée à $(G, C^{wc}_{q_{min}}, D)$. Or nous avons $t_s^{sf}(\alpha_{EDF}, q_{min}) = t_s^{wc}(\alpha_{EDF}, q_{min}) = t_s(\alpha_{EDF})$ et $t_s^{sf}(\alpha, q_{min}) = t_s^{wc}(\alpha, q_{min}) = t_s(\alpha)$. Nous concluons ainsi que :

$$t_s^{sf}(\alpha_{EDF}, q_{min}) \geq t_s^{sf}(\alpha, q_{min}). \quad \square$$

La politique de gestion de qualité sûre utilisée telle qu'elle dans le contrôleur ne conduit pas, en général, à des affectations de qualité régulières, et ce pour deux raisons. D'une part la fonction C^{sf} considère seulement le niveau de qualité q pour la première action, et le niveau de qualité minimal q_{min} pour les autres actions. D'autre part, les durées d'exécution pire-cas ne sont en général pas représentatives du comportement de l'application. Nous allons voir dans la suite comment enrichir la politique de gestion de qualité, en mélangeant C^{sf} avec d'autres fonction de durée d'exécution, afin d'obtenir des politiques de contrôle qui assurent à la fois le respect des échéances, mais aussi la régularité des niveaux de qualité. La proposition suivante est une généralisation de la proposition 3.4. Elle permet de caractériser une classe de politiques de contrôle qui assure le respect des échéances. Les politiques de contrôle proposées dans la suite feront partie de cette classe.

PROPOSITION 3.6 (sûreté) *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain. Soit Γ^X le contrôleur basé sur une politique de contrôle telle que :*

1. *la politique de gestion de qualité est telle que $t_s^X \geq t_s^{sf}$, et ;*
2. *la politique d'ordonnancement est telle que pour toute trace α de G, $\alpha_{q_{min}} = Best\_Sched^X(\alpha, q_{min})$ est un prolongement EDF de α dans G.*

S'il existe un ordonnancement α de G tel que (α, q_{min}) soit un ordonnancement réalisable du système paramétré $SPI(C^{wc})$, alors le contrôleur Γ^X calcule des ordonnancements réalisables du système paramétré incertain $SPI(C)$.

Le résultat précédent est intéressant dans la mesure où il permet de caractériser une classe de politiques de contrôle qui assurent le respect des échéances. Les politiques de contrôle X qui appartiennent à cette classe sont telles que leur politique de gestion de qualité est au moins aussi contraignante que la politique de gestion de qualité sûre, c'est-à-dire $t_s^X \geq t_s^{sf}$, et leur politique d'ordonnancement est la politique d'ordonnancement EDF pour le niveau de qualité minimal q_{min}, c'est-à-dire pour toute trace α nous avons $\alpha_{q_{min}} = Best\_Sched^X(\alpha, q_{min})$ est un prolongement EDF.

La démonstration de la proposition 3.6 est faite en deux temps. Tout d'abord, nous démontrons que le niveau de qualité q_{min} est toujours admissible dans tous les états atteignables du système contrôlé (lemme 3.1). De ce résultat nous déduisons directement que les échéances des actions sont toujours respectées, c'est-à-dire que le contrôleur calcule des ordonnancement réalisables.

LEMME 3.1 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain tel qu'il existe un ordonnancement α de G tel que (α, q_{min}) soit un ordonnancement réalisable de $SPI(C^{wc})$. Soit Γ^X un contrôleur sûr. Alors, le niveau de qualité q_{min} est admissible à n'importe quel état atteignable de $SPI(C)||\Gamma^X$.*

Preuve du lemme 3.1 : Voir annexe. □

Preuve de la proposition 3.6 : D'après le lemme 3.1, nous avons $q_{min} \in \{q \mid t_s^{sf}(\alpha_q, q) \geq t\}$ à n'importe quel état de $SPI(C)||\Gamma$. Soient (α, θ, t) et (α', θ', t') deux états consécutifs de $SPI(C)||\Gamma$ tels que $(\alpha, \theta, t) \stackrel{(a,q_m)}{\rightarrow} (\alpha', \theta', t')$. Soit $\alpha_q = Best\_Sched^{sf}(\alpha, q)$ l'ordonnancement prévu par le contrôleur

pour le niveau de qualité q. Alors, nous avons $t_s^{sf}(\alpha_{q_m}, q_m) \geq t$ et l'action $a = \alpha_{q_m}(1)$ respecte son échéance :

$$\begin{aligned} & t_s^{sf}(\alpha_{q_m}, q_m) \geq t \\ \Rightarrow \quad & \mathbf{min}_{1 \leq k \leq |\alpha_{q_m}|}\, t_s^{sf}(\alpha_{q_m}, q_m)(k) \geq t \\ \Rightarrow \quad & D(a) - C^{wc}(a, q_m) \geq t \\ \Rightarrow \quad & D(a) \geq t + C^{wc}(a, q_m) \geq t'. \end{aligned}$$

Ceci démontre que toutes les actions respectent leur échéance, c'est-à-dire que l'ordonnancement (α, θ) est réalisable. □

D'après la proposition précédente, un contrôleur sûr Γ^X d'un système paramétré incertain $SPI(C) = (G, C^{wc}, D, Q; C)$calcule des ordonnancements réalisables de $SPI(C)$, pour peu qu'il existe un ordonnancement réalisable du système paramétré $SPI(C^{wc})$. Cette dernière hypothèse est une hypothèse de travail minimale étant donné que C^{wc} est une valeur possible du paramètre C. Ainsi, si nous voulons trouver à coup sûr un ordonnancement réalisable pour n'importe quel paramètre C tel que $C \leq C^{wc}$, il est nécessaire qu'il existe un ordonnancement réalisable pour dans le cas particulier $C = C^{wc}$.

3.2.3 Les limites de la politique de contrôle sûre

Le contrôleur Γ^{sf} basé sur la politique de contrôle sûre assure à la fois le respect des échéances et un bon taux d'utilisation des ressources. Nous constatons cependant que les niveaux de qualité choisis par le contrôleur ne sont pas réguliers. En effet, en étant trop permissive, la politique de gestion de qualité sûre conduit le Gestionnaire de Qualité à choisir des niveaux de qualité très élevés pour les premières actions, et donc des niveaux très faibles pour les dernières actions, afin d'assurer le respect des échéances. Dans ce qui suit, nous proposons d'améliorer le contrôleur afin d'obtenir une meilleure la régularité des niveaux de qualité choisis par le contrôleur. Pour ce faire, nous modifions la politique de contrôle dans le but de mieux prévoir le comportement de l'application. Ainsi, la politique de contrôle sera non seulement basée sur une estimation des durées d'exécution pire-cas, mais également sur une estimation des durées d'exécution moyennes.

3.3 Politique de contrôle simple

La politique de contrôle sûre est basée non seulement sur des estimations pire-cas des durées d'exécution, mais également sur des estimations moyennes des durées d'exécution. Ces estimations peuvent être obtenues par analyse statique et/ou par des techniques de *profiling* [44].

3.3.1 Durée d'exécution moyennes

La durée d'exécution pire-cas $C^{wc}(a, q)$ d'une action a pour le niveau de qualité q est une borne supérieure de la durée d'exécution réelle $C(a, q)$. Ainsi, du point de vue du contrôleur, la connaissance

de la fonction C^{wc} permet d'apporter une précision dans la prévision des comportements possibles de l'application, par élimination des comportements dans lesquels $C(a,q) > C^{wc}(a,q)$. Cependant, si les durées d'exécution pire-cas permettent de restreindre l'éventail des comportements possibles, elles ne permettent pas de connaître le comportement le plus probable. L'utilisation des durées d'exécution pire-cas conduit le plus souvent à une double approximation :

- D'une part, les durées d'exécution pire-cas correspondent souvent à des situations très peu probables, c'est-à-dire que les durées d'exécution réelles sont très inférieures aux durées d'exécution pire-cas [48, 49] ;
- D'autre part, il est en général très difficile de calculer statiquement la "vraie" durée d'exécution pire-cas [25]. Ainsi, la plupart du temps, la fonction de durée d'exécution pire-cas C^{wc} ne fournie qu'une borne supérieure des "vraie" durées d'exécution pire-cas. La figure 3.3 illustre ce phénomène en représentant une distribution type des durées d'exécution d'une action pour un niveau de qualité donné.

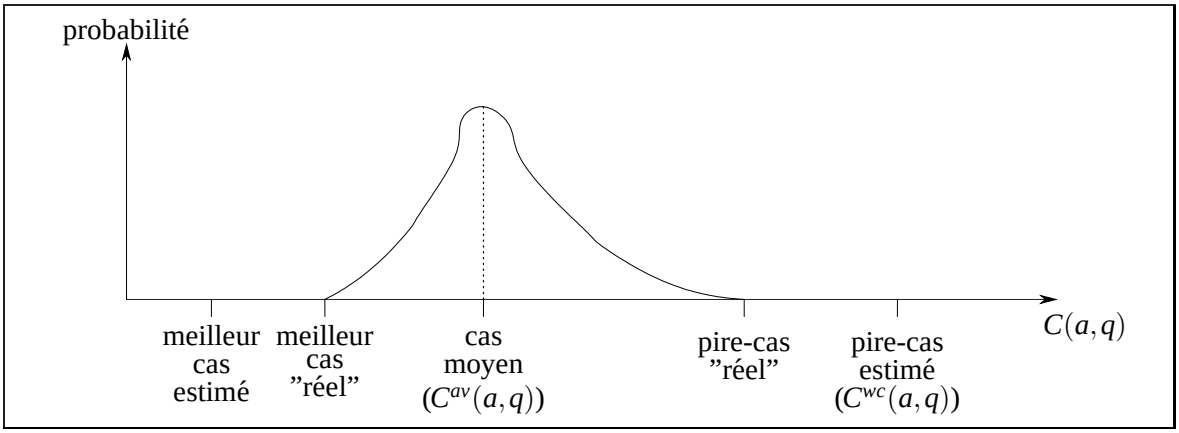


FIG. 3.3: Distribution typique de la durée d'exécution d'une action.

Cette double approximation est évidemment très pénalisante d'un point de vue des performances du contrôleur, et surtout d'un point de vue de la régularité des niveaux de qualité. Nous ne pouvons pas nous passer des durées d'exécution pire-cas pour garantir le respect des échéances, mais il tout de même possible d'utiliser d'autres estimations afin de mieux renseigner le contrôleur. Ainsi, les politiques présentées dans ce qui suit seront non seulement basées sur les durées d'exécution pire-cas, c'est-à-dire C^{wc}, mais également sur des estimations *moyennes*, notées C^{av}. Ces dernières représentent les durées d'exécution "probables" ou "typiques" des actions.

DÉFINITION 3.10 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain. Nous appelons fonction de durée d'exécution* **moyenne** *une fonction du la forme $C^{av} : A \times Q \to \mathbb{R}^+$, telle que :*

1. *$C^{av} \leq C^{wc}$.*
2. *Pour toute action $a \in A$, la fonction $q \mapsto C^{av}(a,q)$ est croissante.*

Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain. Comme nous l'avons fait dans le chapitre 2, nous étendons la fonction du durée d'exécution moyenne $C^{av} : A \times Q \to \mathbb{R}^+$ aux couples

(α,θ) tels que α soit une séquence d'actions et θ soit une affectation de qualité définie sur $\mathsf{ens}(\alpha)$, de la manière suivante :

$$C^{av}(\alpha,\theta) = \sum_{1\leq i\leq|\alpha|} C^{av}(\alpha(i),\theta(\alpha(i))).$$

Nous obtenons ainsi une fonction de durée d'exécution moyenne C^{av} du type $A^*\times\Theta\rightarrow\mathbb{R}^+$. Ceci permet de construire la fonction d'ordonnancement t_s^{av} , et donc la politique de gestion de qualité *moyenne*. Puisque nous n'avons pas forcément $t_s^{av}\leq t_s^{sf}$, la politique de gestion de qualité moyenne n'assure pas que le contrôleur calcule des ordonnancements qui respectent des échéances. Nous allons voir dans la suite comment mélanger les durées d'exécution moyennes et pire-cas afin de satisfaire les trois critères auxquels nous nous intéressons, c'est-à-dire le respect des échéances, une utilisation maximale des ressources de calcul, et une bonne régularité des niveaux de qualité.

PROPOSITION 3.7 *La fonction de durée d'exécution moyenne $C^{av}:A^*\times\Theta\rightarrow\mathbb{R}^+$ est croissante au sens de la définition 3.2. De plus, la politique de gestion de qualité moyenne est conservatrice par rapport aux durées d'exécution moyennes.*

Preuve de la proposition : Le fait que la fonction de durée d'exécution moyenne $C^{av}:A^*\times\Theta\rightarrow\mathbb{R}^+$ soit croissante découle immédiatement du fait que celle-ci est obtenue comme extension de la fonction $C^{av}:A\times Q\rightarrow\mathbb{R}^+$.

Nous allons à présent montrer que la politique de gestion de qualité moyenne est conservatrice par rapport à la fonction de durée d'exécution moyenne, c'est-à-dire :

$$t_s^{av}(\alpha_1\alpha_2,q)\geq t \;\Rightarrow\; t_s^{av}(\alpha_2,q)\geq t+C^{av}(\alpha_1,q).$$

Nous avons :

$$\begin{array}{rcl}
t_s^{av}(\alpha_1\alpha_2,q)\geq t & \Rightarrow & \mathbf{min}_{1\leq k\leq|\alpha_1\alpha_2|}\; t_s^{av}(\alpha_1\alpha_2,q)(k)\geq t\\
& \Rightarrow & \mathbf{min}_{|\alpha_1|+1\leq k\leq|\alpha_1\alpha_2|}\; t_s^{av}(\alpha_1\alpha_2,q)(k)\geq t\\
& \Rightarrow & \mathbf{min}_{1\leq k\leq|\alpha_2|}\; t_s^{av}(\alpha_1\alpha_2)(k+|\alpha_1|)\geq t\\
& \Rightarrow & \mathbf{min}_{1\leq k\leq|\alpha_2|}\; D(\alpha_2(k))-C^{av}(\alpha_1{\alpha_2}^k,q)\geq t\\
& \Rightarrow & \mathbf{min}_{1\leq k\leq|\alpha_2|}\; D(\alpha_2(k))-C^{av}(\alpha_1,q)-C^{av}({\alpha_2}^k,q)\geq t\\
& \Rightarrow & \mathbf{min}_{1\leq k\leq|\alpha_2|}\; t_s^{av}(\alpha_2,q)(k)\geq t+C^{av}(\alpha_1,q)\\
t_s^{av}(\alpha_1\alpha_2,q)\geq t & \Rightarrow & t_s^{av}(\alpha_2,q)\geq t+C^{av}(\alpha_1,q).\quad\square
\end{array}$$

La politique de gestion de qualité moyenne est conservatrice par rapport aux durées d'exécution moyennes. Au contraire, la politique de gestion de qualité sûre n'est, en général, ni conservatrice par rapport à la fonction de durée d'exécution moyenne, ni par rapport à la fonction de durée d'exécution pire-cas. En effet, considérons un système dans lequel, pour toutes les actions $a\in A$ et tous les niveaux de qualité $q\in Q=\{0,1\}$, nous avons $C^{av}(a,q)=C^{wc}(a,q)=q+1$ et $D(a)=3$. Dans ce cas, si a_1 et a_2 sont deux actions, alors $t_s^{sf}(a_1a_2,1)=3-2-1=0$. Or $t_s^{sf}(a_2,1)=3-2=1<2=C(a_1,1)$. Ceci démontre que la politique de gestion de qualité sûre n'est pas conservatrice par rapport à aux fonctions de durée d'exécution C^{av} et C^{wc}.

3.3.2 Politique de gestion de qualité simple

La politique de gestion de qualité simple est construite de façon à prendre en compte à la fois le comportement pire-cas de l'application, mais aussi le comportement moyen de l'application. Cette analyse des comportements possibles de l'application permet à la fois

DÉFINITION 3.11 *Nous appelons fonction de durée d'exécution* simple *la fonction* $C^{sp} : A^* \times \Theta \rightarrow \mathbb{R}^+$ *définie par* $C^{sp} = \mathbf{max}\{ C^{av}, C^{sf} \}$. *La politique de gestion de qualité* **simple** *est alors définie par la fonction d'ordonnancement* t_s^{sp} *associée à* C^{sp}.

PROPOSITION 3.8 *La fonction d'ordonnancement* t_s^{sp} *vérifie* $t_s^{sp} = \mathbf{min}\{ t_s^{av}, t_s^{sf} \}$.

Preuve de la proposition : Nous avons :

$$\begin{aligned}
t_s^{sp}(\alpha,\theta) &= \mathbf{min}_{1\leq k\leq|\alpha|}\ D(\alpha^k) - C^{sp}(\alpha^k,\theta) \\
&= \mathbf{min}_{1\leq k\leq|\alpha|}\ D(\alpha^k) - \mathbf{max}\{ C^{av}(\alpha^k,\theta), C^{sf}(\alpha^k,\theta) \} \\
&= \mathbf{min}_{1\leq k\leq|\alpha|}\ \mathbf{min}\ \{ D(\alpha^k) - C^{av}(\alpha^k,\theta)\ ,\ D(\alpha^k) - C^{sf}(\alpha^k,\theta) \} \\
&= \mathbf{min}\{ (\mathbf{min}_{1\leq k\leq|\alpha|}\ D(\alpha^k) - C^{av}(\alpha^k,\theta))\ ,\ (\mathbf{min}_{1\leq k\leq|\alpha|}\ D(\alpha^k) - C^{sf}(\alpha^k,\theta)) \} \\
t_s^{sp}(\alpha,\theta) &= \mathbf{min}\{ t_s^{av}, t_s^{sf} \}. \quad \square
\end{aligned}$$

A l'instar de la politique de gestion de qualité sûre, la politique de gestion de qualité simple n'est conservatrice ni par rapport à la fonction de durée d'exécution moyenne, ni par rapport à la fonction de durée d'exécution pire-cas. Considérons un système dans lequel, pour toutes les actions $a \in A$ et tous les niveaux de qualité $q \in Q = \{0,1\}$, nous avons $C^{av}(a,q) = q+1$, $C^{wc}(a,q) = 2q+1$ et $D(a) = 4$. Calculons d'abord $t_s^{sp}(a_1a_2,1)$:

$$\begin{aligned}
t_s^{sp}(a_1a_2,1) &= D(a_2) - \mathbf{max}\{ C^{av}(a_1a_2,1), C^{sf}(a_1a_2,1) \} \\
&= 4 - \mathbf{max}\ \{4,4\} = 0.
\end{aligned}$$

Ensuite, nous calculons $t_s^{sp}(a_2,1)$:

$$\begin{aligned}
t_s^{sp}(a_2,1) &= D(a_2) - \mathbf{max}\ C^{av}(a_2,1), C^{sf}(a_2,1) \\
&= 4 - \mathbf{max}\ \{2,3\} = 1 < 2 = C^{av}(a_1,1).
\end{aligned}$$

Ainsi, la politique de gestion de qualité simple n'est pas conservatrice par rapport à la fonction de durée d'exécution moyenne. Puisque $C^{av} \leq C^{wc}$, elle n'est pas non plus conservatrice par rapport à C^{wc}.

3.3.3 Les limitations de la politique simple

La politique de contrôle simple présente l'intérêt de tenir compte à la fois d'un critère de sûreté, du fait de l'utilisation de la fonction des durées d'exécution pire-cas, et d'un critère de régularité

des niveaux de qualité, du fait de l'utilisation des durées d'exécution moyennes. Cependant, elle peut conduire dans certains cas à des affectations de qualité très irrégulières, et en particulier à une réduction du niveau de qualité avant une échéance critique.

Ainsi, le contrôleur Γ^{sp} choisit parfois un niveau de qualité q à un instant donné même s'il n'est pas capable de tenir ce niveau de qualité par la suite, bien que les durées d'exécution réelles soient égales ou même inférieures aux durées d'exécution moyennes. Nous considérons que ce phénomène n'est pas acceptable dans la mesure où, les durées d'exécutions moyennes étant connues d'avance, le contrôleur devrait être capable d'anticiper et choisir des niveaux de qualité réguliers lorsque les durées d'exécution réelles s'approchent des durées moyennes.

Considérons un système paramétré incertain $SPI(C)$ composé de trois actions, quatre niveaux de qualité $Q = \{1,\ldots,4\}$, et une seule échéance $D = 9$ (pour tout $a \in A$, $D(a) = 9$). Nous supposons que les fonctions C, C^{av} et C^{wc} sont des fonctions constantes sur A, et sont données par la figure 3.4. Cette même figure donne les différentes affectations de qualité obtenues par l'implémentation des politiques de contrôle sûre, simple et mixte (définie ci-dessous). Nous remarquons que les niveaux de qualité obtenus en utilisant la politique de gestion de qualité simple sont plus réguliers que les niveaux obtenus en utilisant la politique de gestion de qualité sûre, ce qui correspond bien à ce que nous attendions. Cependant, dans ce cas particulier où les durées d'exécution réelles sont égales aux estimations moyennes, nous nous attendons à une régularité plus importante. Cette amélioration est réalisée par la politique de gestion de qualité mixte que nous présentons dans la section suivante.

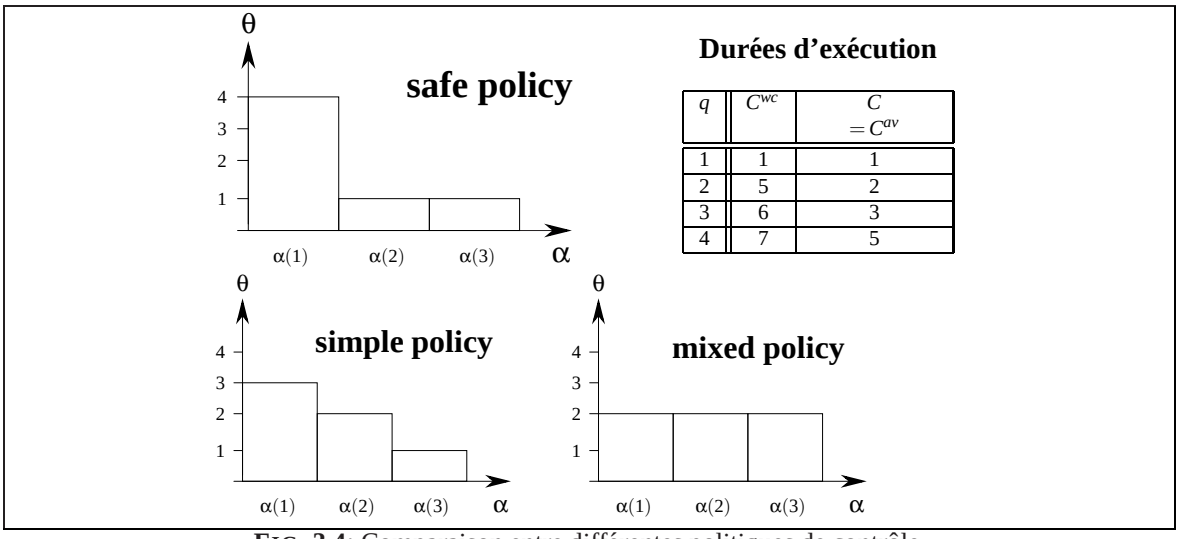


FIG. 3.4: Comparaison entre différentes politiques de contrôle.

3.4 Politique de contrôle mixte

Nous présentons ici la politique de gestion de qualité mixte. A l'instar de la politique de gestion de qualité simple, la politique mixte est construite à partir d'un mélange entre les estimations pire-

cas et les estimations moyennes, et plus précisément entre les fonctions C^{av} et C^{sf}. Cependant, ce mélange est réalisé de façon beaucoup plus fine dans le cas de la politique mixte. La figure 3.5 illustre la différence entre le calcul de la fonction C^{sp} et le calcul de la fonction C^{mx}. Nous obtenons de la sorte une meilleure estimation du comportement de l'application, et donc une plus grande régularité des niveaux de qualité choisis par le contrôleur.

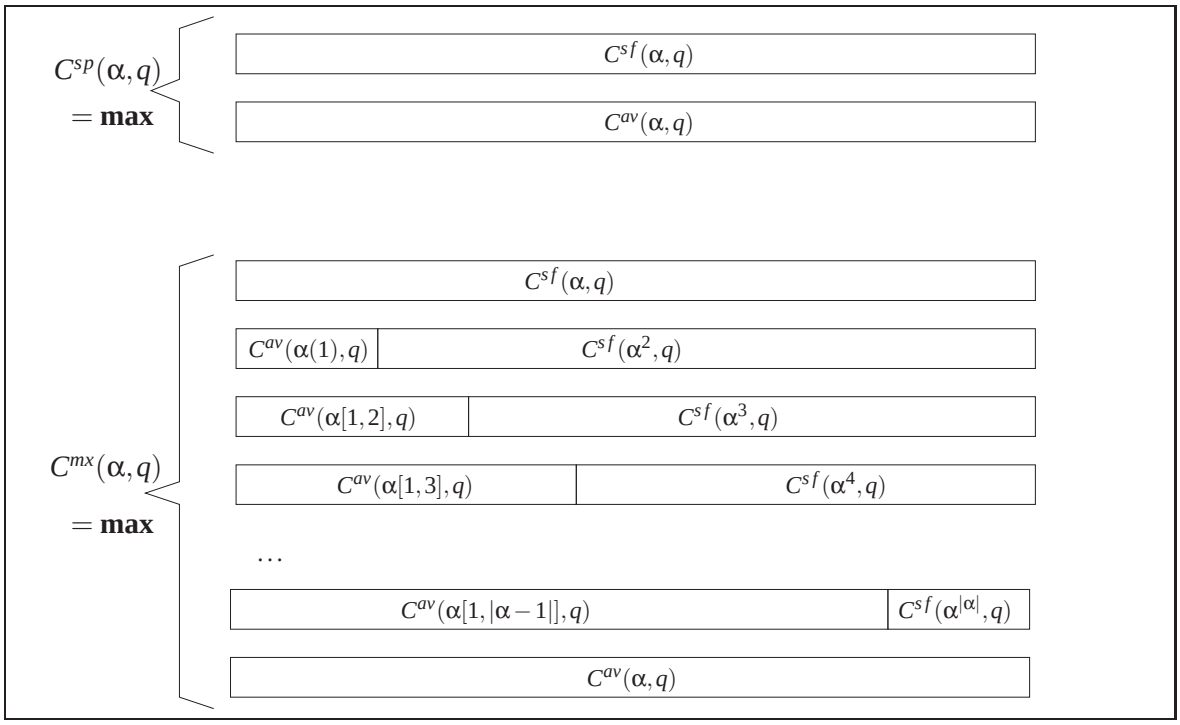


FIG. 3.5: Comparaison entre le calcul de la politique de contrôle mixte et simple.

3.4.1 Politique de gestion de qualité mixte

Pour construire la politique de gestion de qualité mixte, nous introduisons d'abord la fonction de durée d'exécution mixte C^{mx}. Celle-ci permet de construire la fonction d'ordonnancement mixte t_s^{mx} sur laquelle est basée la politique de gestion de qualité mixte.

DÉFINITION 3.12 *Nous appelons fonction de durée d'exécution mixte la fonction partielle* $C^{mx} : A^* \times \Theta \rightarrow \mathbb{R}^+$ *définie pour les couples* (α,θ) *tels que* α *soit une séquence d'actions et* θ *une affectation de qualité définie sur* $\mathsf{ens}(\alpha)$ *de la façon suivante. Nous définissons d'abord la quantité* $C^{mx}(\alpha,\theta)(k)$ *par :*

$$C^{mx}(\alpha,\theta)(k) = C^{av}(^k\alpha,\theta) + C^{sf}(\alpha^{k+1},\theta).$$

Nous définissons pour finir $C^{mx}(\alpha,\theta)$ *par :*

$$C^{mx}(\alpha,\theta) = \mathbf{max}_{0\leq k\leq|\alpha|}\, C^{mx}(\alpha,\theta)(k).$$

Soit (α,θ) un couple pour lequel la fonction de durée d'exécution mixte est définie. La quantité $C^{mx}(\alpha,\theta)$ est calculée comme le maximum entre la durée d'exécution moyenne de (α,θ), c'est-à-dire $C^{av}(\alpha,\theta)$, la durée d'exécution sûre de (α,θ), c'est-à-dire $C^{sf}(\alpha,\theta)$, et toutes les situations intermédiaires, pour lesquelles les k premières actions sont comptées avec la fonction C^{av} et les suivantes sont comptées avec la fonction C^{sf} (voir figure 3.5).

PROPOSITION 3.9 *La fonction de durée d'exécution mixte est croissante au sens de la définition 3.2.*

Preuve de la proposition : Soient α_1 et α_2 deux séquences d'actions, et θ une affectation de qualité définie sur $\mathsf{ens}(\alpha_1\alpha_2)$. Alors :

$$\begin{aligned} C^{mx}(\alpha_1\alpha_2,\theta) &= \mathbf{max}_{1\leq k\leq|\alpha_1\alpha_2|}\, C^{mx}(\alpha_1\alpha_2,\theta)(k) \\ &= \mathbf{max}_{1\leq k\leq|\alpha_1\alpha_2|}\, C^{av}({}^k(\alpha_1\alpha_2),\theta)+C^{sf}((\alpha_1\alpha_2)^{k+1},\theta) \\ &\geq \mathbf{max}_{1\leq k\leq|\alpha_1|}\, C^{av}({}^k(\alpha_1\alpha_2),\theta)+C^{sf}((\alpha_1\alpha_2)^{k+1},\theta) \\ &\geq \mathbf{max}_{1\leq k\leq|\alpha_1|}\, C^{av}({}^k\alpha_1,\theta)+C^{sf}({\alpha_1}^{k+1},\theta). \quad \square \end{aligned}$$

DÉFINITION 3.13 *La politique de gestion de qualité* **mixte** *est définie par la fonction partielle t_s^{mx} : $A^*\times\Theta\rightarrow\mathbb{R}$ associée à la fonction de durée d'exécution mixte C^{mx}.*

La politique de gestion de qualité mixte est bien entendu basée sur la fonction de durée d'exécution mixte. Cette dernière est issue d'un mélange à grain fin entre les fonctions de durée d'exécution moyenne et sûre. Le but de ce mélange est d'anticiper le sur-coût induit par la sûreté, par rapport au comportement moyen de l'application. Ce sur-coût est mesuré par la fonction δ^{max}, dont nous donnons la définition ci-dessous.

DÉFINITION 3.14 *Nous définissons tout d'abord la fonction δ par :*

$$\delta(\alpha,\theta)=C^{sf}(\alpha,\theta)-C^{av}(\alpha,\theta).$$

Nous définissons ensuite la fonction δ^{max} de la façon suivante :

$$\delta^{max}(\alpha,\theta)=\mathbf{max}_{1\leq k\leq|\alpha|}\,\delta(\alpha^k,\theta).$$

EXEMPLE 3.1 *Soit un ordonnancement $\alpha=abcdefg$ et $q>q_{min}$ un niveau de qualité, tels que les durées d'exécution des actions sont données par la figure 3.6. Dans ce cas, les fonctions $k\mapsto\delta(\alpha^k,q)$ et $k\mapsto\delta^{max}(\alpha^k,q)$ sont représentées dans la figure 3.7.*

PROPOSITION 3.10 *La fonction δ^{max} satisfait les propriétés suivantes :*

1. $\delta^{max}\geq 0$.
2. $C^{mx}=C^{av}+\delta^{max}$.

action x	$C^{wc}(x,q) - C^{av}(x,q)$	$C^{wc}(x,q_{min}) - C^{av}(x,q)$
a	100	-10
b	10	-2
c	10	-4
d	80	-10
e	10	-10
f	20	-30
g	10	-10

FIG. 3.6: Durées d'exécution.

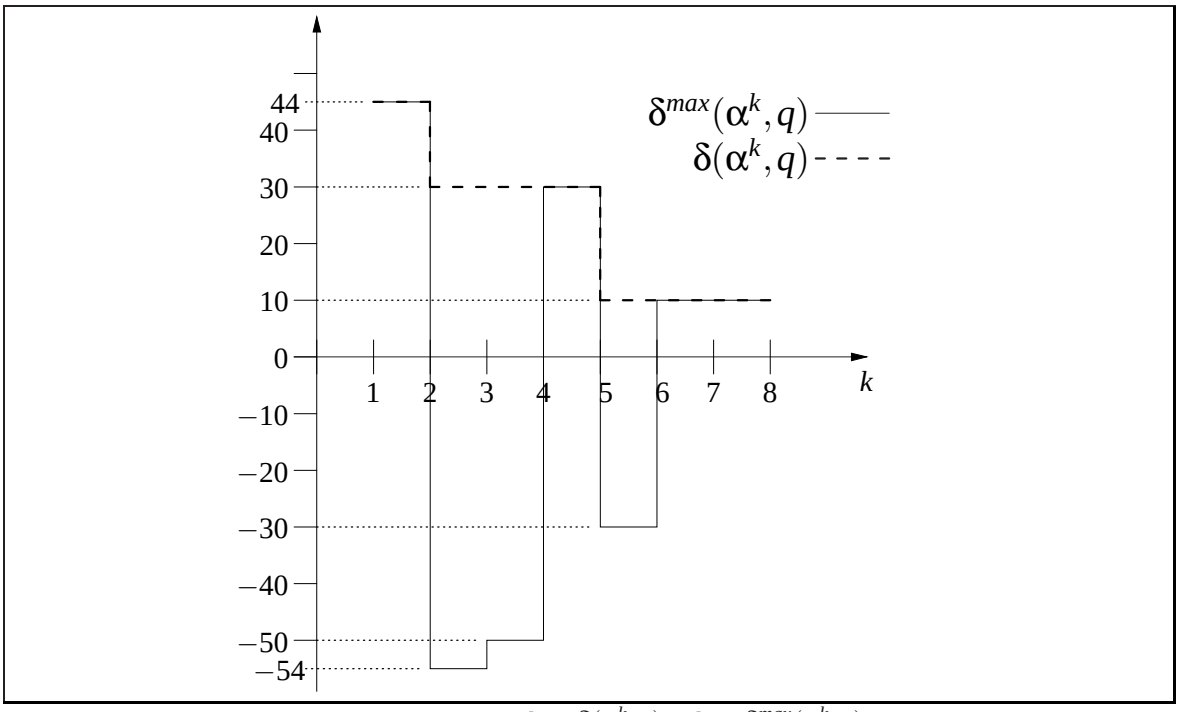


FIG. 3.7: Fonctions $k \mapsto \delta(\alpha^k, q)$ et $k \mapsto \delta^{max}(\alpha^k, q)$.

3. $\delta^{max}(\alpha_1\alpha_2,\theta) \geq \delta^{max}(\alpha_2,\theta)$.

Preuve :

1. Il suffit de voir que $\delta(\alpha,\theta) \geq \delta(\alpha^{|\alpha|},\theta) = C^{wc}(\alpha(|\alpha|,\theta) - C^{av}(\alpha(|\alpha|,\theta) \geq 0$.
2. Nous remarquons que :

$$\begin{aligned} C^{mx}(\alpha,\theta)(k) &= C^{av}({}^k\alpha,\theta) + C^{sf}(\alpha^{k+1},\theta) \\ &= C^{av}({}^k\alpha,\theta) + C^{av}(\alpha^{k+1},\theta) - C^{av}(\alpha^{k+1},\theta) + C^{sf}(\alpha^{k+1},\theta) \\ &= C^{av}(\alpha,\theta) + \delta(\alpha^{k+1},\theta). \end{aligned}$$

Ainsi, $C^{mx}(\alpha,\theta)$ s'écrit :

$$\begin{aligned} C^{mx}(\alpha,\theta) &= \mathbf{max}_{1\leq k\leq|\alpha|}\, C^{mx}(\alpha,\theta)(k) \\ &= \mathbf{max}_{1\leq k\leq|\alpha|}\, C^{av}(\alpha,\theta) + \delta(\alpha^{k+1},\theta) \\ &= C^{av}(\alpha,\theta) + \mathbf{max}_{0\leq k\leq|\alpha|}\, \delta(\alpha^{k+1},\theta) \end{aligned}$$

Puisque $\delta(\alpha^{|\alpha|+1},\theta) = \delta(\varepsilon,\theta) = 0$, nous pouvons conclure que :

$$\begin{aligned} C^{mx}(\alpha,\theta) &= C^{av}(\alpha,\theta) + \mathbf{max}_{1\leq k\leq|\alpha|}\, \delta(\alpha^k,\theta) \\ C^{mx}(\alpha,\theta) &= C^{av}(\alpha,\theta) + \delta^{max}(\alpha,\theta). \end{aligned}$$

3. Nous avons :

$$\begin{aligned} \delta^{max}(\alpha_1\alpha_2,\theta) &= \mathbf{max}_{1\leq k\leq|\alpha_1\alpha_2|}\, \delta((\alpha_1\alpha_2)^k,\theta) \\ &\geq \mathbf{max}_{|\alpha_1|+1\leq k\leq|\alpha_1\alpha_2|}\, \delta((\alpha_1\alpha_2)^k,\theta) \\ &\geq \mathbf{max}_{1\leq k\leq|\alpha_2|}\, \delta({\alpha_2}^k,\theta) \\ \delta^{max}(\alpha_1\alpha_2,\theta) &\geq \delta^{max}(\alpha_2,\theta). \quad \square \end{aligned}$$

La politique de gestion de qualité simple n'est par conservatrice par rapport aux estimations moyennes, ce qui explique le manque de régularité des niveaux de qualité obtenus parfois avec le contrôleur Γ^{sp}. La politique de gestion de qualité mixte est, quant à elle, conservatrice par rapport aux durées d'exécution moyennes. Elle conduira à une meilleure régularité des niveaux de qualité, comme nous le verrons dans le chapitre 5 concernant les résultat expérimentaux.

PROPOSITION 3.11 *La politique de gestion de qualité mixte est conservatrice par rapport à la fonction de durée d'exécution moyenne.*

Preuve de la proposition : Soient α_1 et α_2 deux séquences d'actions, et q un niveau de qualité. Supposons que $t_s^{mx}(\alpha_1\alpha_2,q) \geq t$, et que $C(\alpha_1,q) = C^{av}(\alpha_1,q)$. Nous avons :

$$\begin{aligned} & t_s^{mx}(\alpha_1\alpha_2,q) \geq t \\ \Rightarrow\ & \mathbf{min}_{1\leq k\leq|\alpha_1\alpha_2|}\, D({}^k\alpha_1\alpha_2) - C^{mx}({}^k(\alpha_1\alpha_2),q) \geq t \\ \Rightarrow\ & \mathbf{min}_{1\leq k\leq|\alpha_1\alpha_2|}\, D({}^k\alpha_1\alpha_2) - C^{av}({}^k(\alpha_1\alpha_2),q) - \delta^{max}({}^k(\alpha_1\alpha_2),q) \geq t \\ \Rightarrow\ & \mathbf{min}_{|\alpha_1|+1\leq k\leq|\alpha_1\alpha_2|}\, D({}^k\alpha_1\alpha_2) - C^{av}({}^k(\alpha_1\alpha_2),q) - \delta^{max}({}^k(\alpha_1\alpha_2),q) \geq t \\ \Rightarrow\ & \mathbf{min}_{1\leq k\leq|\alpha_2|}\, D({}^k\alpha_2) - C^{av}(\alpha_1,q) - C^{av}({}^k\alpha_2,q) - \delta^{max}(\alpha_1({}^k\alpha_2),q) \geq t \end{aligned}$$

Puisque $\delta^{max}(\alpha_1(^k\alpha_2)) \geq \delta^{max}(^k\alpha_2)$ et $C^{av}(\alpha_1,q) = C(\alpha,q)$, nous obtenons :

$$\begin{aligned} & \mathbf{min}_{1\leq k\leq|\alpha_2|}\, D(^k\alpha_2) - C(\alpha_1,q) - C^{av}(^k\alpha_2,q) - \delta^{max}(^k\alpha_2,q) \geq t \\ \Leftrightarrow \quad & t_s^{mx}(\alpha_2,q) \geq t + C(\alpha_1,q). \quad \square \end{aligned}$$

3.4.2 Politique d'ordonnancement mixte

Nous avons donné la définition de la politique de gestion de qualité mixte dans la section précédente. Nous étudions ici le problème de l'optimisation de la politique d'ordonnancement. Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain. La politique d'ordonnancement est optimale par rapport à la politique de gestion de qualité mixte si pour toute trace α de G et tout niveau de qualité q, le prolongement $\alpha_q = \mathit{Best\_Sched}^{mx}$ de α dans le graphe G maximise $t_s^{mx}(\alpha_q,q)$ parmi tous les prolongement α' et α (voir section 3.1.2).

Cas d'une échéance unique

Nous nous intéressons en premier lieu au cas où la fonction d'échéance D est une fonction constante. Soit α une trace et $\alpha_q = \mathit{Best\_Sched}^{mx}(\alpha,q)$ le prolongement de α par l'Ordonnanceur considéré. Dans ce cas $\mathsf{critique}_D(\alpha_q) = \{|\alpha_q|\}$. Puisque la politique de gestion de qualité mixte est croissante (proposition 3.9), et d'après la proposition 3.1 nous avons :

$$t_s^{mx}(\alpha_q,q) = \mathbf{max}_{k\in\mathsf{critique}_{D(\alpha_q)}}\, t_s^{mx}(\alpha_q,q)(k) = t_s^{mx}(\alpha_q,q)(|\alpha_q|) = D - C^{mx}(\alpha_q,q).$$

Les résultats de la proposition 3.10 permettent de ré-écrire la fonction C^{mx} à l'aide des fonctions C^{av} et δ^{max} :

$$t_s^{mx}(\alpha_q,q) = D - C^{av}(\alpha_q,q) - \delta^{max}(\alpha_q,q).$$

Puisque la durée moyenne $C^{av}(\alpha_q,q)$ ne dépend que de $\mathsf{ens}(\alpha_q)$, la maximisation de la fonction d'ordonnancement t_s^{mx} revient à une minimisation de la fonction δ^{max}. Nous cherchons ainsi un prolongement $\alpha_q = \mathit{Best\_Sched}^{mx}(\alpha,q)$ tel que pour tout prolongement α' de α nous avons :

$$\delta^{max}(\alpha_q,q) \leq \delta^{max}(\alpha',q).$$

DÉFINITION 3.15 *Nous définissons les fonction* $\eta : A\times Q \to \mathbb{R}^+$ *et* $\beta : A\times Q \to \mathbb{R}$ *de la façon suivante :*

$$\begin{aligned} \eta(a,q) &= C^{wc}(a,q) - C^{av}(a,q) \\ \beta(a,q) &= C^{wc}(a,q_{min}) - C^{av}(a,q). \end{aligned}$$

Pour une action a et un niveau de qualité q, la valeur $\eta(a,q)$ est la différence entre la durée d'exécution pire-cas et la durée moyenne de a au niveau de qualité q. La fonction η peut être vue comme une mesure de l'incertitude sur la durée d'exécution d'une action pour un niveau de qualité

donné. La valeur $\beta(a,q)$, quant à elle, est la différence entre la durée d'exécution pire-cas de a au niveau de qualité minimal, et sa durée moyenne au niveau de qualité q. Ainsi, la fonction β mesure la capacité d'une action à "récupérer" l'application, lorsque des durées d'exécution réelles trop importantes conduisent le contrôleur à accélérer l'application en réduisant le niveau de qualité. Pour une action donnée, plus β est petit et plus l'action permet de récupérer le système. La fonction δ peut être exprimée en fonction des fonctions η et β :

$$\delta(\alpha,\theta) = \eta(\alpha(1),\theta) + \beta(\alpha^2,\theta).$$

La proposition suivante donne trois règles d'échange des actions d'un prolongement α. L'application d'une de ces trois règle permet de baisser la valeur de la fonction δ^{max}. Dans la suite, ces règles serviront de base au calcul d'ordonnancement qui minimisent δ^{max}.

PROPOSITION 3.12 *Soient α un prolongement d'une trace α_0 dans un graphe G, et deux actions a et b indépendantes (voir définition 2.1) et consécutives, c'est-à-dire qu'il existe i tel que $a = \alpha(i)$ et $b = \alpha(i+1)$. Considérons le prolongement α' de α dans lequel les actions a et b ont été échangées, c'est-à-dire $\alpha'(j) = \alpha(j)$ pour tout $j \notin \{i, i+1\}$, $\alpha'(i) = b$, $\alpha'(i) = b$ et $\alpha'(i+1) = a$. Alors, nous avons $\delta^{max}(\alpha',q) \leq \delta^{max}(\alpha,q)$ dans les situations suivantes :*
R1 : $\eta(a,q) \leq \eta(b,q)$, $\beta(a,q) \leq 0$ *et* $\beta(b,q) \leq 0$
R2 : $\beta(a,q) \leq 0$ *et* $\beta(b,q) \geq 0$
R3 : $(\eta-\beta)(a,q) \geq (\eta-\beta)(b,q)$, $\beta(a,q) \geq 0$, *et* $\beta(b,q) \geq 0$.

Preuve de la proposition : Nous savons que $\delta^{max}(\alpha,q) = \mathbf{max}_{1\leq j\leq|\alpha|}\ \delta(\alpha^j,q)$ et $\delta^{max}(\alpha',q) = \mathbf{max}_{1\leq j\leq|\alpha'|}\ \delta(\alpha'^j,q)$. Nous allons donc comparer, pour tout $j \in \{\ 1,\ldots,|\alpha|\ \} = \{\ 1,\ldots,|\alpha'|\ \}$, les valeurs $\delta(\alpha^j,q)$ et $\delta(\alpha'^j,q)$. Soient $I_1 = \{\ i, i+1,\ \}$ et $I_2 = \{\ 1,\ldots,i-1,i+1,\ldots,|\alpha|\ \}$.

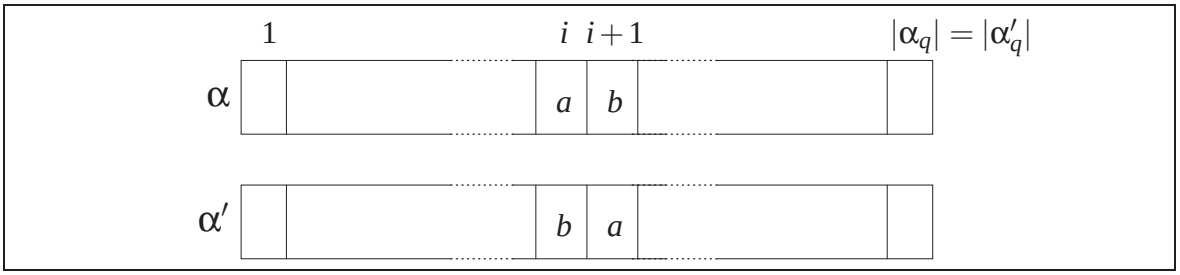


FIG. 3.8: Prolongements α et α'.

Pour tout $j \in I_2$, nous avons :

$$\begin{aligned}\delta(\alpha^j,q) &= \eta(\alpha(j),q) + \beta(\alpha^{j+1},q)\\ &= \eta(\alpha(j),q) + \sum_{k=j+1}^{|\alpha|} \beta(\alpha(k),q).\end{aligned}$$

Puisque $\{ \alpha(j+1),\ldots,\alpha(|\alpha|) \} = \{ \alpha'(j+1),\ldots,\alpha'(|\alpha|) \}$, et $\alpha(j) = \alpha'(j)$ nous obtenons $\delta(\alpha^j,q) = \delta(\alpha'^j,q)$. Il nous reste à comparer les valeurs de δ pour $j \in I_1\{\, i, i+1 \,\}$. Nous avons :

$$\begin{aligned} \delta(\alpha^i,q) &= \eta(a,q)+\beta(b,q)+\beta(\alpha^{i+2},q) \\ \delta(\alpha^{i+1},q) &= \eta(b,q)+\beta(\alpha^{i+2},q). \end{aligned}$$

Puisque $\{ \alpha(i+2),\ldots,\alpha(|\alpha|) \} = \{ \alpha'(i+2),\ldots,\alpha'(|\alpha|) \}$, nous avons :

$$\begin{aligned} \delta(\alpha'^i,q) &= \eta(b,q)+\beta(a,q)+\beta(\alpha^{i+2},q) \\ \delta(\alpha'^{i+1},q) &= \eta(a,q)+\beta(\alpha^{i+2},q). \end{aligned}$$

Puisque $\delta^{max}(\alpha,q) = \mathbf{max}_{j\in I_1\cup I_2}\ \delta(\alpha,q)$ et $\delta^{max}(\alpha',q) = \mathbf{max}_{j\in I_1\cup I_2}\ \delta(\alpha',q)$, il suffit de montrer que $\mathbf{max}_{j\in I_1}\ \delta(\alpha',q) \leq \mathbf{max}_{j\in I_1}\ \delta(\alpha',q)$ pour assurer $\delta^{max}(\alpha',q) \leq \delta^{max}(\alpha,q)$. Les résultats suivants découlent directement des définitions :

$$\beta(a,q) \leq 0 \ \Rightarrow\ \delta(\alpha'^i,q) \leq \delta(\alpha^{i+1},q) \tag{3.1}$$

$$\beta(b,q) \geq 0 \ \Rightarrow\ \delta(\alpha'^{i+1},q) \leq \delta(\alpha^i,q) \tag{3.2}$$

$$\eta(a,q) \leq \eta(b,q) \ \Rightarrow\ \delta(\alpha'^{i+1},q) \leq \delta(\alpha^{i+1},q) \tag{3.3}$$

R1 : Supposons que a et b sont tels que $\beta(a,q) \leq 0$, $\beta(b,q) \leq 0$ et $\eta(a,q) < \eta(b,q)$. Puisque $\beta(a,q) \leq 0$ et $\eta(a,q) < \eta(b,q)$, nous avons $\delta(\alpha'^i,q) \leq \delta(\alpha^{i+1},q) \leq \mathbf{max}_{j\in I_1}\ \delta(\alpha^j,q)$ et $\delta(\alpha'^{i+1},q) \leq \delta(\alpha^{i+1},q) \leq \mathbf{max}_{j\in I_1}\ \delta(\alpha^j,q)$ (implications 3.1 et 3.3). Ainsi, nous obtenons $\mathbf{max}_{j\in I_1}\ \delta(\alpha'^j,q) \leq \mathbf{max}_{j\in I_1}\ \delta(\alpha^j,q)$. Nous pouvons conclure que, dans ce cas, nous avons $\delta^{max}(\alpha',q) \leq \delta^{max}(\alpha,q)$.

R2 : Puisque $\beta(a,q) \leq 0$ et $\beta(b,q) > 0$, nous avons $\delta(\alpha'^i,q) \leq \delta(\alpha^{i+1},q) \leq \mathbf{max}_{j\in I_1}\ \delta(\alpha^j,q)$ et $\delta(\alpha'^{i+1},q) \leq \delta(\alpha^i,q) \leq \mathbf{max}_{j\in I_1}\ \delta(\alpha^j,q)$ (implications 3.1 et 3.2). Ainsi, nous obtenons $\mathbf{max}_{j\in I_1}\ \delta(\alpha'^j,q) \leq \mathbf{max}_{j\in I_1}\ \delta(\alpha^j,q)$. Nous pouvons conclure que, dans ce cas, nous avons $\delta^{max}(\alpha',q) \leq \delta^{max}(\alpha,q)$.

R3 : Supposons que a et b sont telles que $\beta(a) > 0$, $\beta(b) > 0$, et $(\eta-\beta)(a) > (\eta-\beta)(b)$. Puisque $\beta(b,q) > 0$, nous avons $\delta(\alpha'^{i+1},q) \leq \delta(\alpha^i,q) \leq \mathbf{max}_{j\in I_1}\ \delta(\alpha^j,q)$ (implication 3.2). De plus, puisque $(\eta-\beta)(a,q) > (\eta-\beta)(b,q)$ nous avons :

$$\begin{aligned} \delta(\alpha'^i,q)-\delta(\alpha^i,q) &= \eta(a,q)+\beta(b,q)-(\eta(b,q)+\beta(a,q)) \\ &= (\eta-\beta)(a,q)-(\eta-\beta)(b,q) \\ \delta(\alpha'^i,q)-\delta(\alpha^i,q) &\leq 0. \end{aligned}$$

Ainsi, nous obtenons $\delta(\alpha'^i,q) \leq \delta(\alpha^i,q) \leq \mathbf{max}_{j\in I_1}\ \delta(\alpha^j,q)$. Ainsi, nous obtenons $\mathbf{max}_{j\in I_1}\ \delta(\alpha'^j,q) \leq \mathbf{max}_{j\in I_1}\ \delta(\alpha^j,q)$. Nous pouvons conclure que, dans ce cas, nous avons $\delta^{max}(\alpha',q) \leq \delta^{max}(\alpha,q)$. □

La proposition suivante résout le problème de la recherche d'un ordonnancement qui minimise δ^{max} pour un niveau de qualité q donné, dans cas d'un graphe de précédence G sans arcs de précédence, c'est-à-dire $G = (A,\emptyset)$. Un tel ordonnancement est obtenu en partitionnant les actions

selon la valeur de la fonction β. En effet, toutes les actions a telles que $\beta(a,q) > 0$ sont ordonnancées en premier, et les actions a telles que $\beta(a,q) \leq 0$ sont ordonnancées en dernier. De plus, les actions a telles que $\beta(a,q) > 0$ sont ordonnancées selon $\eta(a,q)$ décroissant. Les actions a telles que $\beta(a,q) \leq 0$ sont ordonnancées selon $(\eta - \beta)(a,q)$ croissant. Ce principe est illustré par la figure 3.9, et est formalisé dans la proposition suivante.

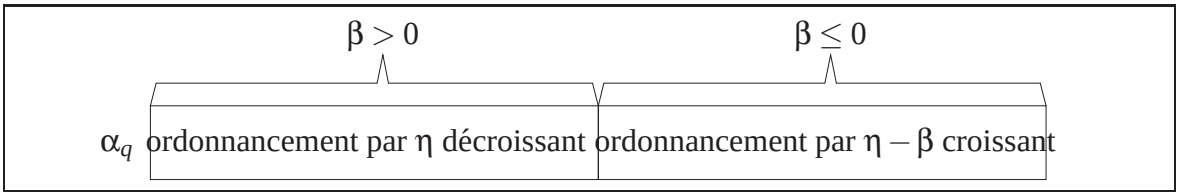


FIG. 3.9: Ordonnancement minimisant δ^{max} pour un niveau de qualité q.

PROPOSITION 3.13 (ordonnancement d'un graphe sans arcs) *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain tel que le graphe de précédence $G = (A, \prec)$ n'ait pas d'arc, c'est-à-dire $\prec = \emptyset$. Soient un prolongement α_q d'une trace α, un niveau de qualité q, et un indice i tel que :*

1. *Pour tout $j \in \{ 1, \ldots, i \}$ nous avons $\beta(\alpha_q(j), q) > 0$, et pour tout $j \in \{ i+1, \ldots, |\alpha| \}$ nous avons $\beta(\alpha_q(j), q) \leq 0$.*
2. *Pour tout $j_1, j_2 \in \{ 1, \ldots, i \}$, $j_1 < j_2 \Rightarrow \eta(\alpha_q(j_1), q) \geq \eta(\alpha_q(j_2), q)$.*
3. *Pour tout $j_1, j_2 \in \{ i+1, \ldots, |\alpha| \}$, $j_1 < j_2 \Rightarrow (\eta - \beta)(\alpha_q(j_1), q) \leq (\eta - \beta)(\alpha_q(j_2), q)$.*

Alors le prolongement α_q minimise δ^{max}, c'est-à-dire que pour tout prolongement α'_q de α nous avons $\delta^{max}(\alpha_q, q) \leq \delta^{max}(\alpha'_q, q)$.

Preuve de la proposition : Soit α'_q un prolongement quelconque de la trace α. Puisque le graphe de précédence ne comporte pas d'arc, il est possible d'obtenir le prolongement α_q à partir de α'_q en utilisant ré-ordonnançant les actions de α'_q à l'aide des règles de ré-ordonnancement R1, R2, R3 de la proposition 3.12. Puisqu'à chaque ré-ordonnancement la valeur δ^{max} diminue, nous pouvons conclure que $\delta^{max}(\alpha_q, q) \leq \delta^{max}(\alpha'_q, q)$. □

PROPOSITION 3.14 (ordonnancement dans le cas $\beta \leq 0$) *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain. Soit q un niveau de qualité tel pour toute action $a \in A$, $\beta(a,q) \leq 0$. Nous définissons la fonction η^* comme suit :*

$$\eta^*(a,q) = \mathbf{max}_{a'=a \,\vee\, a \prec a'} \, \eta(a', q).$$

Soit un prolongement α_q d'une trace α tel que pour tout $i < j$ nous avons $\eta^*(\alpha_q(i), q) \geq \eta^*(\alpha_q(j), q)$. Alors le prolongement α_q minimise δ^{max}, c'est-à-dire que pour tout prolongement α'_q de α nous avons $\delta^{max}(\alpha_q, q) \leq \delta^{max}(\alpha'_q, q)$.

LEMME 3.2 *Soient $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain, q un niveau de qualité tel que $\beta(a,q) \leq 0$ pour tout $a \in A$, et α une trace de G. Pour tout prolongement α_q de α, nous avons :*

$$\delta^{max*}(\alpha_q, q) = \delta^{max}(\alpha_q, q),$$

où δ^{max*} *est calculé comme* δ^{max} *en remplaçant les valeurs* η *par* η^**, c'est-à-dire :*

$$\delta^{max*}(\alpha_q,q) = \mathbf{max}_{1\leq i\leq|\alpha_q|}\ \delta^*({\alpha_q}^i,q) = \mathbf{max}_{1\leq i\leq|\alpha_q|}\ \eta^*(\alpha_q(i),q)+\beta({\alpha_q}^{i+1},q).$$

Preuve du lemme : Nous allons montrer tout d'abord $\delta^{max}(\alpha_q,q) \geq \delta^{max*}(\alpha_q,q)$. Puisque $\delta^{max*}(\alpha_q,q) = \mathbf{max}_{1\leq j\leq|\alpha_q|}\ \delta^*({\alpha_q}^j,q)$, il existe $i \in \{\ 1,\ldots,|\alpha_q|\ \}$ tel que $\delta^*({\alpha_q}^i,q) = \delta^{max*}(\alpha_q,q)$. Nous posons $a = \alpha_q(i)$. Nous avons $\delta^*({\alpha_q}^i,q)) = \eta^*(a,q)+\beta(\alpha^{i+1},q)$. Par définition, nous avons $\eta^*(a,q) = \mathbf{max}_{a'=a\ \vee\ a\prec a'}\ \eta(a',q)$. Autrement dit, il existe une action a' telle que $a' = a$ ou $a \prec a'$, et telle que $\eta(a',q) = \eta^*(a,q)$. Puisque α_q est un prolongement, il existe forcément $j \geq i$ tel que $\alpha_q(j) = a'$. Puisque $\eta(a',q) = \eta^*(a,q)$ nous obtenons :

$$\begin{aligned}\delta({\alpha_q}^j,q) &= \eta(a',q)+\beta({\alpha_q}^{j+1},q)\\ \delta({\alpha_q}^j,q) &= \eta^*(a,q)+\beta({\alpha_q}^{j+1},q).\end{aligned}$$

Puisque $\beta \leq 0$, nous avons :

$$\begin{aligned}\delta({\alpha_q}^j,q) &\geq \eta^*(a,q)+\beta(\alpha_q[i+1,j],q)+\beta({\alpha_q}^{j+1},q)\\ \delta({\alpha_q}^j,q) &\geq \delta^*({\alpha_q}^i,q) = \delta^{max*}({\alpha_q}^i,q)\\ \delta^{max}(\alpha_q,q) \geq \delta({\alpha_q}^j,q) &\geq \delta^{max*}({\alpha_q}^i,q).\end{aligned}$$

Nous avons donc montré $\delta^{max}(\alpha_q,q) \geq \delta^{max*}({\alpha_q}^i,q)$.

Nous montrons à présent que $\delta^{max*}(\alpha_q,q) \geq \delta^{max}(\alpha_q,q)$. Nous pourrons alors conclure que $\delta^{max*}(\alpha_q,q) = \delta^{max*}(\alpha_q,q)$. D'après la définition de η^*, nous avons immédiatement, pour toute action a et tout niveau de qualité q, $\eta^*(a,q) \geq \eta(a,q)$. Ainsi, pour tout $i \in \{\ 1,\ldots,|\alpha_q|\ \}$, nous avons $\delta^*({\alpha_q}^i,q) \geq \delta({\alpha_q}^i,q)$, ce qui démontre $\delta^{max*}(\alpha_q,q) \geq \delta^{max}(\alpha_q,q)$. □

Preuve de la proposition 3.14 : En appliquant la règle R1 de la proposition 3.12 aux fonctions δ^{max*} et η^*, nous montrons que le prolongement α_q dans lequel les actions sont ordonnancées par η^* décroissant maximise δ^{max*}, et donc δ^{max} d'après le résultat du lemme précédent. □

PROPOSITION 3.15 (ordonnancement dans le cas $\beta \geq 0$**)** *Soit* $SPI(C) = (G,C^{wc},D,Q;C)$ *un système paramétré incertain. Soit q un niveau de qualité tel pour toute action* $a \in A$, $\beta(a,q) \geq 0$. *Nous définissons la fonction* $(\eta-\beta)^*$ *comme suit :*

$$(\eta-\beta)^*(a,q) = \mathbf{max}_{a'=a\ \vee\ a'\prec a}\ (\eta-\beta)(a',q).$$

Soit un prolongement α_q *d'une trace* α *tel que pour tout* $i < j$ *nous avons* $(\eta-\beta)^*(\alpha_q(i),q) \leq (\eta-\beta)^*(\alpha_q(j),q)$. *Alors le prolongement* α_q *minimise* δ^{max}*, c'est-à-dire que pour tout prolongement* α'_q *de* α *nous avons* $\delta^{max}(\alpha_q,q) \leq \delta^{max}(\alpha'_q,q)$.

La preuve de la proposition précédente repose sur le lemme suivant.

LEMME 3.3 *Soient* $SPI(C) = (G,C^{wc},D,Q;C)$ *un système paramétré incertain, q un niveau de qualité tel que* $\beta(a,q) \geq 0$ *pour tout* $a \in A$*, et* α *une trace de G. Pour tout prolongement* α_q *de* α*, nous avons :*

$$\delta^{max*}(\alpha_q,q) = \delta^{max}(\alpha_q,q),$$

où δ^{max*} est calculé comme δ^{max} en remplaçant les valeurs η par $(\eta-\beta)^*+\beta$, c'est-à-dire :

$$\delta^{max*}(\alpha_q,q) = \mathbf{max}_{1\leq i\leq|\alpha_q|}\ \delta^*({\alpha_q}^i,q) = \mathbf{max}_{1\leq i\leq|\alpha_q|}\ ((\eta-\beta)^*+\beta)(\alpha_q(i))+\beta({\alpha_q}^{i+1},q).$$

Preuve du lemme :

Nous allons montrer tout d'abord $\delta^{max}(\alpha_q,q) \geq \delta^{max*}(\alpha_q,q)$. Puisque $\delta^{max*}(\alpha_q,q) = \mathbf{max}_{1leqj\leq|\alpha_q|}\ \delta^*({\alpha_q}^j,q)$, il existe $i\in\{\ 1,\ldots,|\alpha_q|\ \}$ tel que $\delta^*({\alpha_q}^i,q) = \delta^{max*}(\alpha_q,q)$. Nous posons $a=\alpha(i)$. Nous avons :

$$\delta^*({\alpha_q}^i,q) = ((\eta-\beta)^*+b)(a,q)+\beta(\alpha^{i+1},q).$$

Par définition, $(\eta-\beta)^*(a,q) = \mathbf{max}_{a'=a\ \vee\ a'\prec a}\ (\eta-\beta)(a',q)$. Ainsi, il existe a' telle que $a'=a$ ou $a'\prec a$, et telle que $(\eta-\beta)(a',q) = (\eta-\beta)^*(a,q)$. Puisque α_q est un prolongement il existe forcément $j\leq i$ tel que $\alpha_q(j)=a'$. Puisque $(\eta-\beta)(a',q) = (\eta-\beta)^*(a,q)$:

$$\begin{array}{rcl}
\delta({\alpha_q}^j,q) & = & \eta(a',q)+\beta({\alpha_q}^{j+1},q)\\
\delta({\alpha_q}^j,q) & = & \eta(a',q)-\beta(a',q)+\beta(a',q)+\beta({\alpha_q}^{j+1},q)\\
\delta({\alpha_q}^j,q) & = & (\eta-\beta)^*(a,q)+\beta(a',q)+\beta({\alpha_q}^{j+1},q)\\
\delta({\alpha_q}^j,q) & = & (\eta-\beta)^*(a,q)+\beta(a',q)+\beta(\alpha_q[j+1,i-1],q)+\beta(a,q)+\beta({\alpha_q}^{i+1},q).
\end{array}$$

Puisque $\beta\geq 0$, nous avons :

$$\begin{array}{rcl}
\delta({\alpha_q}^j,q) & \geq & (\eta-\beta)^*(a,q)+\beta(a,q)+\beta({\alpha_q}^{i+1},q)\\
\delta({\alpha_q}^j,q) & \geq & \delta^*({\alpha_q}^i,q) = \delta^{max}(\alpha_q,q)\\
\delta^{max}(\alpha_q,q) \geq \delta({\alpha_q}^j,q) & \geq & \delta^*({\alpha_q}^i,q).
\end{array}$$

Ce qui prouve $\delta^{max}(\alpha_q,q)\geq\delta^{max*}(\alpha_q,q)$.

Nous montrons à présent que $\delta^{max*}(\alpha_q,q)\geq\delta^{max}(\alpha_q,q)$. Nous pourrons alors conclure que $\delta^{max*}(\alpha_q,q) = \delta^{max*}(\alpha_q,q)$. Le résultat découle immédiatement des définitions. En effet, pour tout a nous avons $(\eta-\beta)^*(a,q)\geq(\eta-\beta)(a,q)$. Nous en déduisons que pour tout i :

$$\begin{array}{rl}
& (\eta-\beta)^*(\alpha_q(i),q)\geq(\eta-\beta)(\alpha_q(i),q)\\
\Rightarrow & (\eta-\beta)^*(\alpha_q(i),q)+\beta(\alpha_q(i),q)+\beta({\alpha_q}^{i+1},q)\geq(\eta-\beta)(\alpha_q(i),q)+\beta(\alpha_q(i),q)+\beta({\alpha_q}^{i+1},q)\\
\Rightarrow & \delta^*({\alpha_q}^i,q)\geq\delta({\alpha_q}^i,q).
\end{array}$$

D'où le résultat $\delta^{max*}(\alpha_q,q)\geq\delta^{max}(\alpha_q,q)$, ce qui termine la démonstration de $\delta^{max*}(\alpha_q,q) = \delta^{max*}(\alpha_q,q)$. □

Preuve de la proposition 3.15 : En appliquant la règle R3 de la proposition 3.12 aux fonctions δ^{max*} et $(\eta-\beta)^*$, nous montrons que le prolongement α_q dans lequel les actions sont ordonnancées par $(\eta-\beta)^*$ croissant maximise δ^{max*}, et donc δ^{max} d'après le résultat du lemme précédent. □

Cas général

Le calcul d'un ordonnanceur *Best_Sched*^{mx} optimal par rapport à la politique de gestion de qualité mixte est un problème difficile. Une première heuristique consiste à restreindre l'exploration des prolongements aux prolongements EDF. Nous allons voir comment utiliser les résultats obtenus dans le paragraphe précédent afin de optimiser un ordonnanceur *Best_Sched*^{mx} qui se limite aux prolongements EDF.

DÉFINITION 3.16 *Soient $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain, α une trace de G, et α_q un prolongement de α. Nous dirons que le prolongement α est* **EDF-optimal** *par rapport au niveau de qualité q si :*

1. *α_q est un prolongement EDF de α, c'est-à-dire un ordonnancement EDF de $G/(A \setminus \mathsf{ens}(\alpha))$ par rapport à D.*
2. *α_q maximise t_s^{mx} parmi les prolongements EDF, c'est-à-dire :*

$$t_s^{mx}(\alpha_q, q) = \mathbf{max}\ \{\ t_s^{mx}(\alpha', q) \mid \alpha' \in EDF(G/(A \setminus \mathsf{ens}(\alpha)), D)\ \}.$$

Nous dirons qu'un ordonnanceur Best_Sched^{mx} est **EDF-optimal** *par rapport à la politique de gestion de qualité mixte si pour toute trace α de G et tout niveau de qualité q, le prolongement $\alpha_q = Best\_Sched^{mx}(\alpha, q)$ est EDF-optimal par rapport au niveau de qualité q.*

PROPOSITION 3.16 *Soient $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain, α une trace de G et α_q un prolongement EDF de α. Si pour tout l $\alpha_{q,l}$ est un ordonnancement optimal de G/A_l par rapport au niveau de qualité q, alors le prolongement α_q est EDF-optimal par rapport au niveau de qualité q.*

Preuve : Soit $A_1 \ldots A_l$ la partition induite par D^*, $\alpha = \alpha_1 \ldots \alpha_L$ et $\alpha' = \alpha'_1 \ldots \alpha'_L$ deux ordonnancements EDF tels que, pour tout $l \in \{1, \ldots, L\}$, α_l et α'_l sont des ordonnancements de G/A_l. Supposons que, pour tout l, α_l est un ordonnancement EDF-optimal par rapport au niveau de qualité q. En particulier nous avons $\delta^{max}(\alpha_l, q) \leq \delta^{max}(\alpha'_l, q)$.

Soit k_l la longueur de l'ordonnancement $\alpha_1 \ldots \alpha_l$, c'est-à-dire $k_l = |\alpha_1 \ldots \alpha_l|$. Nous avons $t_s^{mx}(\alpha, q) = \mathbf{min}_{1 \leq l \leq L}\ t_s^{mx}(\alpha, q)(k_l)$, et :

$$\begin{aligned}
& t_s^{mx}(\alpha, q)(k_l) = t_s^{av}(\alpha, q)(k_l) - \delta^{max}(\alpha_1 \ldots \alpha_l, q) \\
= \ & t_s^{av}(\alpha, q)(k_l) - \delta^{max}(\alpha_1 \ldots \alpha_l, q) \\
= \ & t_s^{av}(\alpha, q)(k_l) - \mathbf{max}_{1 \leq i \leq k_l}\ \delta((\alpha_1 \ldots \alpha_l)^i, q) \\
= \ & t_s^{av}(\alpha, q)(k_l) - \mathbf{max}_{1 \leq j \leq l}\ \delta^{max}(\alpha_j, q) + \beta(\alpha_{j+1} \ldots \alpha_l, q).
\end{aligned}$$

De plus, puisque $\beta(\alpha_{i+1} \ldots \alpha_l, q) = \beta(\alpha'_{i+1} \ldots \alpha'_l, q)$ et $\delta^{max}(\alpha_j, q) \leq \delta^{max}(\alpha'_j, q)$ pour tout j, nous avons :

$$\begin{aligned}
& \mathbf{max}_{1 \leq j \leq l}\ \delta^{max}(\alpha_j, q) + \beta(\alpha_{j+1} \ldots \alpha_l, q) \\
= \ & \mathbf{max}_{1 \leq j \leq l}\ \delta^{max}(\alpha_j, q) + \beta(\alpha'_{j+1} \ldots \alpha'_l, q) \\
\leq \ & \mathbf{max}_{1 \leq j \leq l}\ \delta^{max}(\alpha'_j, q) + \beta(\alpha'_{j+1} \ldots \alpha'_l, q).
\end{aligned}$$

Puisque $t_s^{av}(\alpha,q)(k_l) = t_s^{av}(\alpha',q)(k_l)$ nous avons, pour tout l, $t_s^{mx}(\alpha,q)(k_l) \leq t_s^{mx}(\alpha',q)(k_l)$. Nous pouvons ainsi conclure que $t_s^{mx}(\alpha,q) \leq t_s^{mx}(\alpha',q)$. □

3.5 Diagramme des vitesses

Le diagramme des vitesses est une représentation graphique de l'espace d'état de l'application contrôlée. Nous introduisons la notion de vitesse, qui est le ratio entre un temps théorique basé sur les durées d'exécution moyennes, et le temps-réel. Pour un état donné d'un système paramétré incertain, les choix du gestionnaire de qualité s'interprètent alors en terme de comparaison entre la vitesse optimale, qui correspond à une utilisation optimale des ressources de calcul à partir de l'état courant, et les vitesses idéales, qui correspondent aux différents niveaux de qualité. Le diagramme des vitesses est ainsi scindé en régions.

3.5.1 Interprétation de la politique de gestion de qualité moyenne

Soit $SPI(C) = (G, C^{wc}, D, Q; C)$ un système paramétré incertain, et Γ^{av} un contrôleur de $SPI(C)$ basé sur la politique de contrôle moyenne. Soit (α,θ,t) un état de $SPI(C)||\Gamma^{av}$. Soit $q \in Q$ un niveau de qualité, et $\alpha_q = Best\_Sched^{av}(\alpha,q)$ le prolongement α dans G prévu par la politique d'ordonnancement pour le niveau de qualité q. Soit k l'indice de l'échéance critique du prolongement α_q, c'est-à-dire l'échéance $D(\alpha_q(k))$.

Nous allons tracer le diagramme des vitesses correspondant à l'échéance $D(\alpha_q(k))$ et au niveau de qualité q. Il s'agit d'une représentation graphique en deux dimensions des états possibles du système, et des choix de contrôle en fonction de ces états. L'abscisse du diagramme correspond à l'avancée réelle du temps, ou temps-réel t. Nous avons forcément $D(\alpha_q(k)) \geq t$. L'ordonnée du diagramme correspond à un temps théorique, basé sur le modèle des durées d'exécution moyennes. Ce temps théorique permet de mesurer l'avancée de l'application dans les calculs. Nous effectuons une normalisation par rapport à l'échéance $D(\alpha_q(k))$, de telle sorte que le diagramme ait une forme de carré de côté $D(\alpha_q(k))$. Plus précisément, l'état (α,θ,t) correspond à la position $(t, y(q))$ dans le diagramme des vitesses, où $y(q)$ est donné par :

$$y(q) = D(\alpha_q(k)) \frac{C^{av}(\alpha,q)}{C^{av}(\alpha(^k\alpha_q),q)}.$$

Ainsi, le diagramme des vitesse relie l'avancée réelle du temps à l'avancée théorique du temps : lorsque que le point $(t, y(q))$ est au dessus de la diagonale, nous considérons que l'application est en avance ; si au contraire le point est en dessous de la diagonale, l'application est jugée en retard.

DÉFINITION 3.17 *Nous appelons* **vitesse** *entre deux points du diagramme (t_1, y_1) et (t_2, y_2) la pente du vecteur qui relie ces deux points, c'est-à-dire :*

$$v = \frac{y_2 - y_1}{t_2 - t_1}.$$

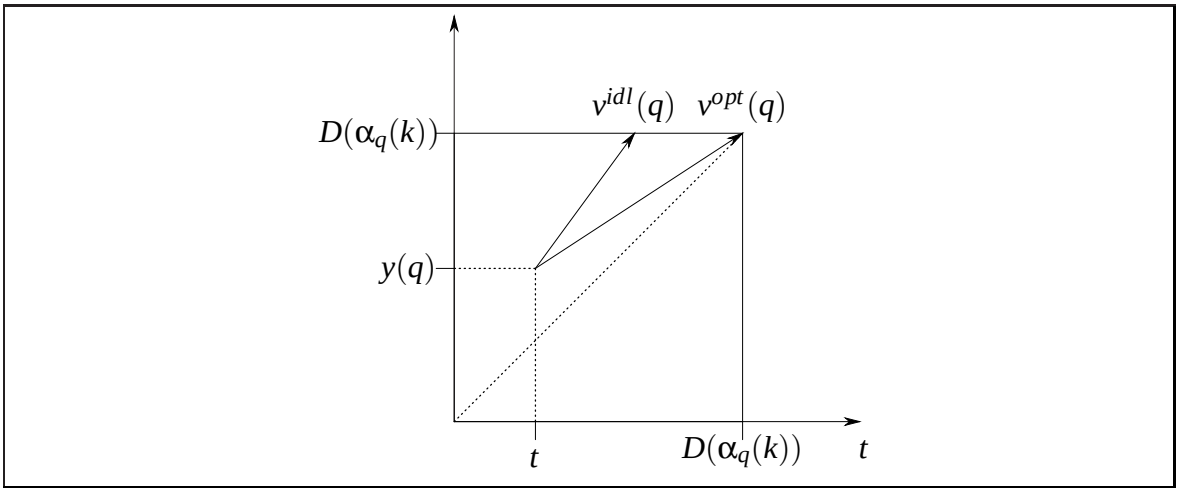


FIG. 3.10: Diagramme des vitesses pour le niveau de qualité q, l'échéance $D(\alpha_q(k))$, et la position $(t, y(q))$.

Pour un niveau de qualité q et une position $(t, y(q))$ dans le diagramme des vitesses, la vitesse idéale $v^{idl}(q)$ correspond à la vitesse du système dans le cas où les durées d'exécution réelles sont exactement les durées d'exécution moyennes C^{av}. Pour un niveau de qualité q donné, la vitesse idéale $v^{idl}(q)$ ne dépend pas de la position courante $(t, y(q))$, mais uniquement de l'échéance critique considérée. Ainsi, tant que l'échéance critique ne change pas, les vitesses idéales restent les même. Elles correspondent à des valeurs intrinsèques du modèle de l'application.

DÉFINITION 3.18 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain. Nous appelons* **vitesses idéales** *à l'état (α, θ, t) de $SPI(C)$, par rapport à l'échéance $D(\alpha_q(k))$, les vitesses $v^{idl}(q)$ définies pour tout niveau de qualité q par :*

$$v^{idl}(q) = \frac{D(\alpha_q(k))}{C^{av}(\alpha(^k\alpha_q), q)}.$$

Pour un niveau de qualité q et une position $(t, y(q))$ dans le diagramme des vitesses, la vitesse optimale $v^{opt}(q)$ est obtenue comme la vitesse entre la position courante $(t, y(q))$ et le point de coordonnées $(D(\alpha_q(k)), D(\alpha_q(k)))$. La figure 3.10 donne un exemple de diagramme des vitesses, dans lequel les vitesses idéales et optimales ont été représentées pour le niveau de qualité q. Il s'agit donc d'un comportement optimal dans la mesure où il correspond à une qualité constante et une utilisation maximale des ressources, c'est-à-dire que l'action $\alpha_q(k)$ termine exactement à la date $D(\alpha_q(k))$ qui est son échéance.

DÉFINITION 3.19 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain. Nous appelons* **vitesses optimales** *à l'état (α, θ, t) de $SPI(C)$, par rapport à l'échéance $D(\alpha_q(k))$, les vitesses $v^{opt}(q)$ définies pour tout niveau de qualité q par :*

$$v^{opt}(q) = \frac{D(\alpha_q(k)) - y(q)}{D(\alpha_q(k)) - t} = \frac{D(\alpha_q(k))}{C^{av}(\alpha(^k\alpha_q), q)} \frac{C^{av}(^k\alpha_q, q)}{D(\alpha_q(k)) - t}.$$

Puisque $D(\alpha_q(k))$ est l'échéance critique du prolongement α_q, nous avons $t_s^{av}(\alpha_q,q) = t_s^{av}(\alpha_q,q)(k)$. La condition d'admissibilité $t^{av}(\alpha_q,q)(k) \geq t$ associée à la politique de gestion de qualité moyenne peut s'exprimer simplement en terme de comparaison entre la vitesse idéale $v^{idl}(q)$ et la vitesse optimale $v^{opt}(q)$ associée. Plus précisément, la condition d'admissibilité est vraie si et seulement si la vitesse idéale $v^{idl}(q)$ est supérieure ou égale à la vitesse optimale associée au niveau $v^{opt}(q)$. Ce résultat est intéressant dans la mesure où il permet de traduire les choix de niveau de qualité obtenus avec la politique de gestion de qualité moyenne, directement dans le diagramme des vitesses. Par exemple, dans la figure 3.10, l'état du système correspondant à la position $(t,y(q))$ est tel que le niveau de qualité q est admissible. En effet, sur cette figure, la vitesse idéale $v^{idl}(q)$ est supérieure à la vitesse optimale $v^{opt}(q)$.

PROPOSITION 3.17 *Soit (α,θ,t) un état d'un système paramétré incertain $SPI(C) = (G,C^{wc},D,Q;C)$. Soient $\alpha_q = Best\_Sched^{av}(\alpha,q)$ les prolongements de α dans G prévus par l'Ordonnanceur, et k l'indice de l'échéance $D(\alpha_q(k))$ dans le prolongement α_q. Alors la position $(t,y(q))$ et les vitesses optimales v^{opt} et idéales $v^{idl}(q)$ associées à l'état (α,θ,t) sont telles que :*

$$v^{idl}(q) \geq v^{opt}(q) \iff t_s^{av}(\alpha_q,q)(k) \geq t.$$

Preuve de la proposition : Nous avons :

$$\begin{aligned}
v^{idl}(q) \geq v^{opt}(q) &\iff \frac{D(\alpha_q(k))}{C^{av}(\alpha({}^k\alpha),q)} \geq \frac{D(\alpha_q(k))}{C^{av}(\alpha({}^k\alpha),q)} \frac{C^{av}({}^k\alpha,q)}{D(\alpha_q(k))-t} \\
&\iff 1 \geq \frac{C^{av}({}^k\alpha,q)}{D(\alpha_q(k))-t} \\
&\iff D(\alpha_q(k)) - C^{av}({}^k\alpha,q) \geq t \\
&\iff t_s^{av}(\alpha_q,q)(k) \geq t. \quad \square
\end{aligned}$$

Dans le cas d'une politique d'ordonnancement statique, et tant que l'échéance critique ne change pas, la proposition 3.17 nous permet de représenter l'ensemble des positions $(t,y(q))$ telles que le niveau de qualité q soit admissible par rapport à la politique de gestion de qualité moyenne. Pour tracer cet ensemble dans le diagramme des vitesses, il suffit de tracer la demi-droite qui passe par le point $(0,0)$, et dont la pente est donnée par le vecteur de vitesse idéale $v^{idl}(q)$. Ainsi, dans la figure 3.11, la zone grisée correspond états du système pour lesquelles le niveau de qualité q n'est pas admissible. Au contraire, la zone en clair ainsi que la demi-droite de pente $v^{idl}(q)$ correspond aux états du système pour lesquels le niveau de qualité q est admissible.

Découpage du diagramme des vitesses en régions

La diagramme des vitesses doit être construit sur chaque état de $SPI(C)||\Gamma^{av}$, et pour chaque niveau de qualité. Nous considérons ici un cas particulier dans lequel nous pourrons comparer les différents états du système et les différents niveaux de qualité sur un diagramme des vitesses unique.

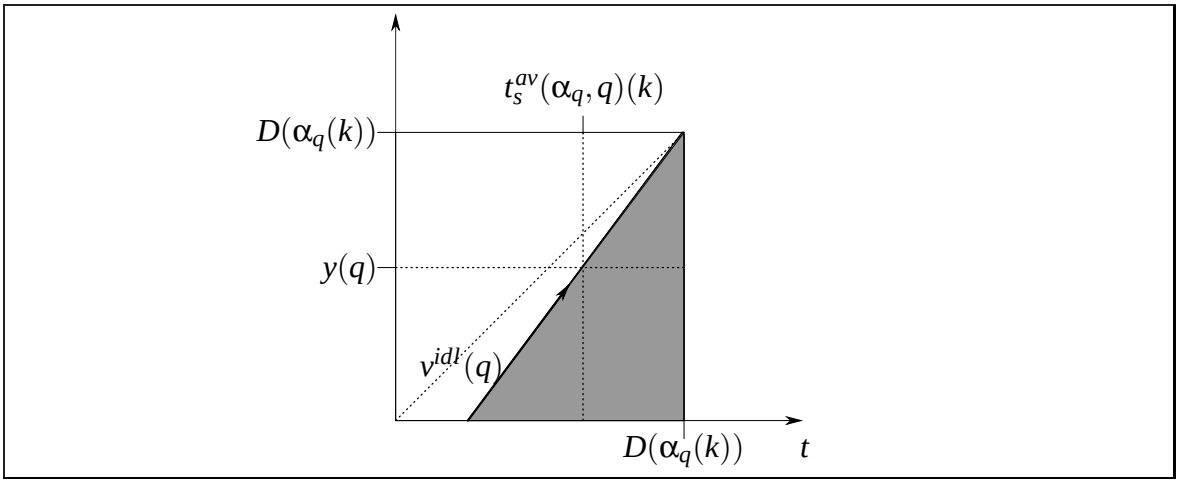


FIG. 3.11: Diagramme des vitesses pour le niveau de qualité q et l'échéance critique $D(\alpha_q(k))$.

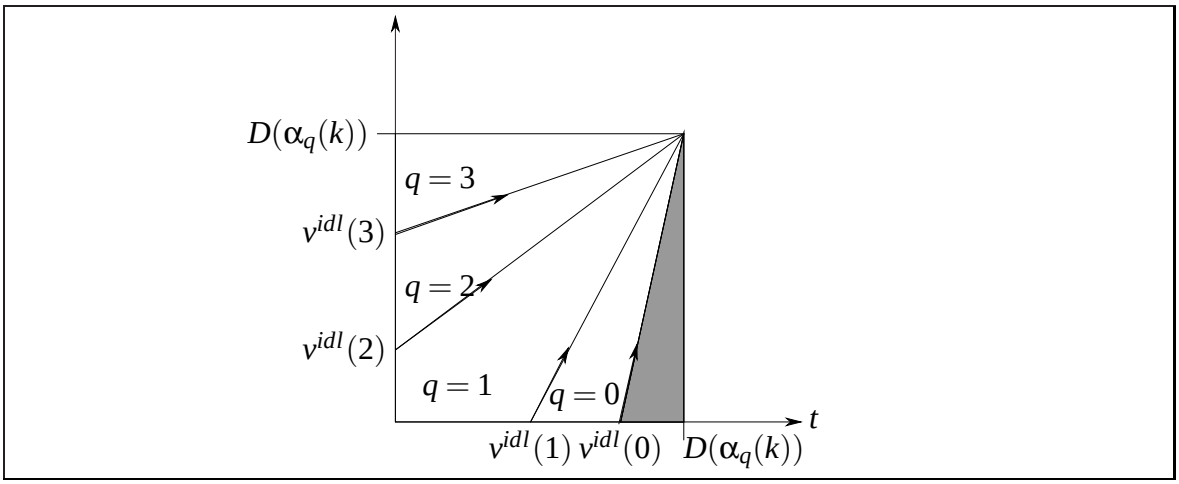


FIG. 3.12: Diagramme de vitesses pour des positions $(t, y(q))$ indépendantes du niveau de qualité q.

Considérons un système paramétré incertain $SPI(C) = (G, C^{wc}, D, Q; C)$, $G = (A, \prec)$, et un contrôleur Γ^{av} basé une politique d'ordonnancement statique telle que, pour tout état (α, θ, t) de $SPI(C)||\Gamma^{av}$, la position dans le diagramme des vitesses $(t, y(q))$ associée à cet état ne dépendent pas de q, c'est-à-dire que $q \mapsto y(q)$ est une fonction constante. Nous notons alors $y(q) = y$. Nous supposons également que la fonction $D : A \to \mathbb{R}^+$ est constante. Ainsi, l'échéance critique est toujours la même, pour tous les états du système (α, θ, t) et pour tous les niveaux de qualité q : il s'agit dans tous les cas de l'échéance de la dernière action des prolongements calculés par la politique d'ordonnancement.

Connaissant un état du système contrôlé $SPI(C)||\Gamma^{av}$, il devient alors possible déterminer une position unique (t, y) dans le diagramme des vitesses. Sur cet état, et donc à la position (t, y), le contrôleur choisit un unique niveau de qualité. Nous pouvons alors découper la diagramme des vitesses en régions, correspondant aux différents niveaux de qualité. La proposition 3.17 nous permet de construire ces régions : leurs frontières sont linéaires et de pente données par les vecteurs de vitesse idéales $v^{idl}(q)$. Ainsi, dans la figure 3.12, la zone grisée correspond aux états pour lesquels aucun niveau de qualité n'est admissible par rapport à la politique de gestion de qualité moyenne. Les autres zones correspondent aux différents niveaux de qualité, à savoir que la frontière supérieure gauche de la zone du niveau de qualité q n'appartient à cette zone, mais à celle du niveau de qualité $q+1$.

Dans le chapitre suivant, nous introduisons la notion de *boucle*. Nous verrons que l'ordonnancement le plus simple d'une boucle est de la forme $\{\alpha_0\}^n = \alpha_0\alpha_0 \ldots \alpha_0$, où α_0 est répété n fois. Or les positions $(t, y(q))$ associées aux états (α, θ, t) tels que α soit un "multiple" de α_0, c'est-à-dire $\alpha = \{\alpha_0\}^m$, sont telles que $y(q)$ est indépendant du niveau de qualité q. Les résultats étudiés ici seront donc pertinents dans ce cas particulier d'une boucle contrainte par une fonction d'échéance D constante.

3.5.2 Interprétation de la politique de gestion de qualité mixte

L'utilisation des durées d'exécution pire-cas dans la politique de gestion de qualité mixte permet d'assurer le respect des échéances. Nous allons voir comment modifier le diagramme des vitesses afin que la condition d'admissibilité $t_s^{mx}(\alpha_q, q) \geq t$ de la politique de gestion de qualité mixte se traduise de façon simple en terme de comparaison entre vitesses. Pour ce faire, nous modifions la vitesse optimale de la manière suivante.

Soit (α, θ, t) un état de $SPI(C)||\Gamma^{mx}$, et $\alpha_q = Best\_Sched^{mx}(\alpha, q)$ les ordonnancements prévus par la politique d'ordonnancement pour les différents niveaux de qualité. Soit q un niveau de qualité, et k l'indice dans α_q de l'échéance critique. La définition des vitesses idéales $v^{idl}(q)$ ainsi que la position courante $(t, y(q))$ ne change pas par rapport à ce qui a été donné pour la politique de contrôle moyenne, dans la section précédente. Nous définissons les *vitesses optimales sûres* de la façon suivante.

DÉFINITION 3.20 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, un système paramétré incertain. Nous appelons* **vitesses optimales sûres** *à l'état (α, θ, t) de $SPI(C)$, par rapport à l'échéance $D(\alpha_q(k))$, les vitesses $v^{opt-sf}(q)$ définies pour tout niveau de qualité q par :*

$$v^{opt-sf}(q) = \frac{D(\alpha_q(k)}{C^{av}(\alpha(^k\alpha), q)} \frac{C^{av}(^k\alpha, q)}{D(\alpha_q(k)) - \delta^{max}(\alpha_q, q) - t}.$$

La vitesse optimale sûre est calculée comme la vitesse entre la position courante $(t, y(q))$, et le point $(D(\alpha_q(k)) - \delta^{max}(\alpha_q, q), D(\alpha_q(k)))$. La figure 3.13 représente un exemple de diagramme des vitesses dans le cas de la politique de contrôle mixte. La proposition suivante donne une interprétation de la politique de gestion de qualité mixte en terme de comparaison entre les vitesses idéales définie dans la section précédente, et les vitesses optimales sûres définies ci-dessus.

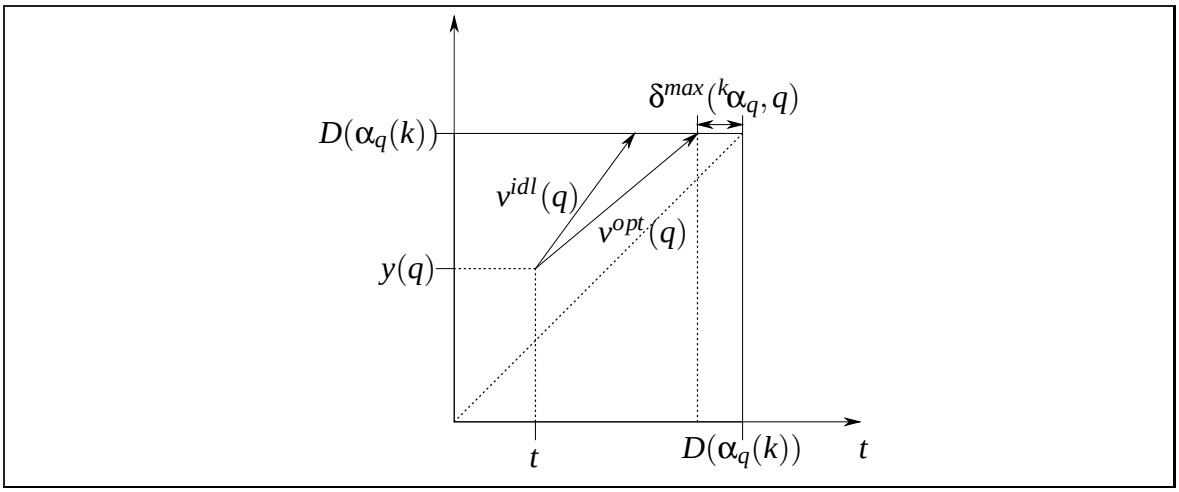


FIG. 3.13: Diagramme des vitesses dans le cas de la politique mixte.

PROPOSITION 3.18 *Soit* (α, θ, t) *un état d'un système paramétré incertain* $SPI(C) = (G, C^{wc}, D, Q; C)$. *Soient* $\alpha_q = Best\_Sched^{av}(\alpha, q)$ *les prolongements de* α *dans* G *prévus par l'Ordonnanceur, et* k *l'indice de l'échéance* $D(\alpha_q(k))$ *dans le prolongement* α_q. *Alors la position* $(t, y(q))$ *et les vitesses optimales* v^{opt-sf} *et idéales* $v^{idl}(q)$ *associées à l'état* (α, θ, t) *sont telles que :*

$$v^{idl}(q) \geq v^{opt-sf}(q) \iff t_s^{mx}(\alpha_q, q)(k) \geq t.$$

Preuve de la proposition : La preuve est similaire à celle qui a été donnée pour la proposition 3.17 :

$$\begin{aligned}
v^{idl}(q) \geq v^{opt}(q) &\iff \frac{D(\alpha_q(k))}{C^{av}(\alpha({}^k\alpha), q)} \geq \frac{D(\alpha_q(k))}{C^{av}(\alpha({}^k\alpha), q)} \frac{C^{av}({}^k\alpha, q)}{D(\alpha_q(k)) - \delta^{max}(\alpha_q, q) - t} \\
&\iff 1 \geq \frac{C^{av}({}^k\alpha, q)}{D(\alpha_q(k)) - \delta^{max}(\alpha_q, q) - t} \\
&\iff D(\alpha_q(k)) - C^{av}({}^k\alpha, q) - \delta^{max}(\alpha_q, q) \geq t \\
&\iff D(\alpha_q(k)) - C^{mx}({}^k\alpha, q) \geq t \\
&\iff t_s^{mx}(\alpha_q, q)(k) \geq t. \quad \square
\end{aligned}$$

Ainsi, l'interprétation de la politique de gestion de qualité mixte en terme de vitesses dans le diagramme des vitesses est similaire à l'interprétation que nous avons donné dans le cas de la politique

de gestion de qualité moyenne. La seule différence se situe au niveau de la vitesse optimale. Dans le cas de la politique mixte, nous utilisons en effet la vitesses optimale sûre, qui ne correspond pas à une utilisation totale des ressources, mais de $D(\alpha_q(k)) - \delta^{max}(\alpha_q, q)$ unités de temps sur les $D(\alpha_q(k))$ disponibles. Ainsi, plus la valeur $\delta^{max}(\alpha_q, q)$ sera petite, et plus l'utilisation des ressources sera proche de l'utilisation maximale. De la sorte, nous faisons apparaître le fait que le respect des échéances limite inévitablement la maximisation du budget de temps. Dans le cas de la politique de contrôle mixte, le contrôleur "prévoit" de perdre $\delta^{max}(\alpha_q, q)$ unités de temps, si les durées d'exécution réelles sont identiques aux durées d'exécution moyennes. Bien entendu, selon la valeur des durées d'exécution réelles, la perte réelle peut être différente de cette valeur.

Découpage du diagramme des vitesses en régions

Comme nous l'avons fait pour la politique de contrôle moyenne, nous allons nous intéresser à un cas particulier de système paramétré incertain avec lequel le diagramme des vitesses va pouvoir être découpé en régions dont les frontières linéaires.

Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, $G = (A, \prec)$ un système paramétré incertain tel qu'il existe un ordonnancement de la forme $\{\alpha_0\}^n$. Nous supposons également que la fonction d'échéance D est constante. Considérons le contrôleur Γ^{mx} utilisant une politique d'ordonnancement statique basée sur $\{\alpha_0\}^n$. Nous ne représentons dans le diagramme des vitesses que les points qui correspondent à des états de la forme $(\{\alpha_0\}^m, \theta, t)$, $m \leq n$. Dans ce cas, la position $(t, y(q))$ associée à l'état $(\{\alpha_0\}^m, \theta, t)$ est indépendante du niveau de qualité q. En effet, nous avons :

$$y(q) = D\frac{C^{av}(\{\alpha_0\}^m, q)}{C^{av}(\{\alpha_0\}^n, q} = D\frac{m}{n}.$$

Un point $(t, y(q))$ correspondant à l'état $(\{\alpha_0\}^m, \theta, t)$ appartient à la région du niveau de qualité q si et seulement si nous avons :

$$t_s^{mx}(\{\alpha_0\}^{n-m}, q) \geq t > t_s^{mx}(\{\alpha_0\}^{n-m}, q+1) \text{ si } q < q_{max},$$

et

$$t_s^{mx}(\{\alpha_0\}^{n-m}, q) \geq t \text{ sinon.}$$

Les équations des frontières des régions sont donc de la forme $t_s^{mx}(\{\alpha_0\}^{n-m}, q) = t$, c'est-à-dire, puisque la fonction d'échéance D est constante, $C^{av}(\{\alpha_0\}^{n-m}, q) + \delta^{max}(\{\alpha_0\}^{n-m}, q) = t$, ou plus simplement $(n-m)C^{av}(\alpha_0, q) + \delta^{max}(\{\alpha_0\}^{n-m}, q) = t$.

Dans la section 4.5.1, nous montrons en particulier que, pour un niveau de qualité q donné, $\delta^{max}(\{\alpha_0\}^{n-m}, q)$ est une fonction linéaire de l'entier m. Ainsi, l'équation d'une frontière d'une région est linéaire en fonction de m. Puisque m est linéaire en fonction de $y(q)$, nous avons démontré que les frontières des régions sont linéaires dans le diagramme des vitesses. La figure 3.14 illustre ce résultat.

Résultat sur l'utilisation du budget de temps

Intuitivement, nous voyons dans le diagramme des vitesse que la valeur δ^{max} est caractéristique dans la politique de gestion de qualité mixte. En effet, elle mesure la différence entre le comportement

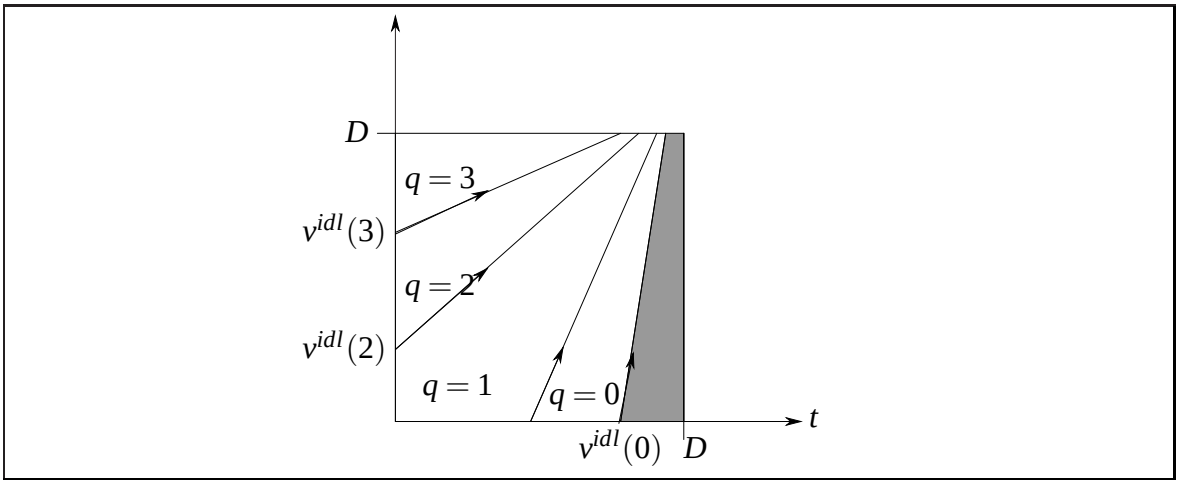


FIG. 3.14: Diagramme de vitesses pour des positions $(t, y(q))$ indépendantes du niveau de qualité q.

pire-cas et le comportement moyen, et a donc un lien avec l'incertitude qui existe sur les durées d'exécution réelles. Nous allons voir que deux facteurs limite l'utilisation du budget de temps. Le premier est justement l'incertitude sur les durées d'exécution ; il sera mesuré grâce à la fonction δ^{max}. Le second concerne la granularité de contrôle. En effet, plus la granularité de contrôle est fine et plus le contrôleur sera en mesure de maximiser le budget de temps en s'approchant au plus près des échéances. Dans le diagramme, ceci s'interprète par une meilleure approximation de la vitesse idéale par les niveaux de qualité discrets du système paramétré incertain.

PROPOSITION 3.19 *Soit $SPI(C) = (G, C^{wc}, D, Q; C)$, $G = (A, \prec)$ un système paramétré incertain tel que sa fonction d'échéance D soit constante. Considérons le contrôleur Γ^{mx} basée sur la politique de gestion de qualité mixte et une politique d'ordonnancement statique basée sur l'ordonnancement α de G. Supposons que l'ordonnancement (α, θ) de $SPI(C)$ calculé par le contrôleur Γ^{mx} soit tel que $q = \theta(\alpha(1)) < q_{max}$, et que les durées d'exécution réelles C soient telles que $C = C^{av}$. Dans ce cas, nous avons le résultat suivant sur l'utilisation du budget de temps disponible :*

$$D - \Delta - \delta^{max}(\alpha, q+1) \leq C(\alpha, \theta) \leq D,$$

où Δ est une valeur caractérisant la granularité de contrôle, définie par :

$$\Delta = \textbf{max} \{ C^{av}(a, \textbf{min}\ ) - C^{av}(a, q) \mid q < q_{max}, a \in A \}.$$

La preuve de la proposition précédente est donnée dans l'annexe.

CHAPITRE 4

Mise en œuvre efficace

DANS le chapitre précédent, nous avons présenté une technique de contrôle à grain fin d'application temps-réel. En agissant sur certains paramètres — les niveaux de qualité — nous permettons à l'application d'utiliser au mieux les ressources de calcul, le but étant de maximiser la qualité de service finale par rapport à des contraintes temps-réel de type échéance. Le contrôle de l'application est obtenu par l'insertion de points de contrôle dans l'application, sur lesquels le code correspondant au contrôleur est appelé afin de choisir la prochaine action à exécuter et fixer son niveau de qualité.

Le contrôleur doit donc s'attacher à la fois à effectuer des choix de contrôle pertinents, mais aussi à ne pas grever inutilement le budget de temps alloué, de façon à pouvoir activer des niveaux de qualité les plus élevés possibles. Les performances du contrôleur dépendent donc non seulement de la technique de contrôle qui est employée, mais aussi de la manière dont celle-ci est implémentée. Par exemple, dans l'implémentation actuelle, nous avons fait le choix de n'utiliser que des politiques d'ordonnancement statique.

Dans ce qui suit, nous abordons le problème de la génération automatique d'un contrôleur à partir du modèle de l'application. Ce travail est basé sur l'implémentation d'un prototype d'outil réalisé au cours de cette thèse. Nous traiterons de la majeure partie des problèmes rencontrés au cours cette implémentation, et en particulier les points suivants :

- Fonctionnement général de l'outil. Nous verrons comment celui-ci s'intègre au sein d'un processus classique de compilation.
- Modélisation de l'application temps-réel. Nous proposons une extension au modèle de système

paramétré incertain présenté dans le chapitre 3. Nous verrons comment construire le modèle d'une application réelle — un encodeur vidéo — dans la chapitre 5.
– Implémentation d'une politique d'ordonnancement statique pour la politique de contrôle simple et la politique de contrôle mixte présentées dans le chapitre 3.
– Implémentation efficace du calcul des fonctions d'ordonnancement correspondant à la politique de gestion de qualité simple et la politique de gestion de qualité mixte. En particulier, nous chercherons un compromis entre le pré-calcul statique d'informations, qui a tendance à augmenter la consommation mémoire et la taille de l'exécutable, et le calcul dynamique, qui augmente la durée d'exécution du contrôleur, et donc de diminuer la qualité de service finale.

4.1 Généralités sur l'outil

Le flot de compilation et de génération de l'application contrôlée est illustré par la figure 4.1. Pour nous, l'application qui se trouve en entrée du flot est complètement décrite avec :

– un système paramétré incertain ; la notion de graphe de précédence hiérarchique nous permettra de générer un contrôleur pour des applications qui contiennent des boucles dont le nombre d'itérations est connu à l'exécution seulement, ce qui est très souvent le cas dans les applications multimédia ;
– une bibliothèque de fonctions écrites en C ; celle-ci fournie une implémentation des actions du système paramétré incertain.

A partir de cette description, l'outil génère (statiquement) un ordonnancement de l'application ainsi qu'un ensemble de tables qui sont utilisées, à l'exécution, pour accélérer l'exécution du code correspondant au contrôleur. Ces tables dépendent bien entendu de la politique de gestion de qualité mise en œuvre. Nous verrons en particulier comment implémenter efficacement un contrôleur basé sur la politique de gestion de qualité simple et sur la politique de gestion de qualité mixte.

4.1.1 Entrées du flot de génération de l'application contrôlée

Les entrées du flot de génération de l'application contrôlée sont de deux types.

Modèle de l'application. Il s'agit d'un système paramétré incertain dont le graphe de précédence est *hiérarchique*. Un tel graphe permet de modéliser la répétition d'un sous-ensemble d'actions, ce qui est particulièrement utile pour modéliser les applications multimédia qui sont souvent constituées d'algorithmes itératifs.

Bibliothèque de fonctions. Ces fonctions, écrites en langage C dans l'implémentation actuelle, décrivent l'effet de l'exécution des actions. Dans l'implémentation de l'outil, le lien entre une action et sa fonction correspondante peut être assuré par une table fournie en entrée.

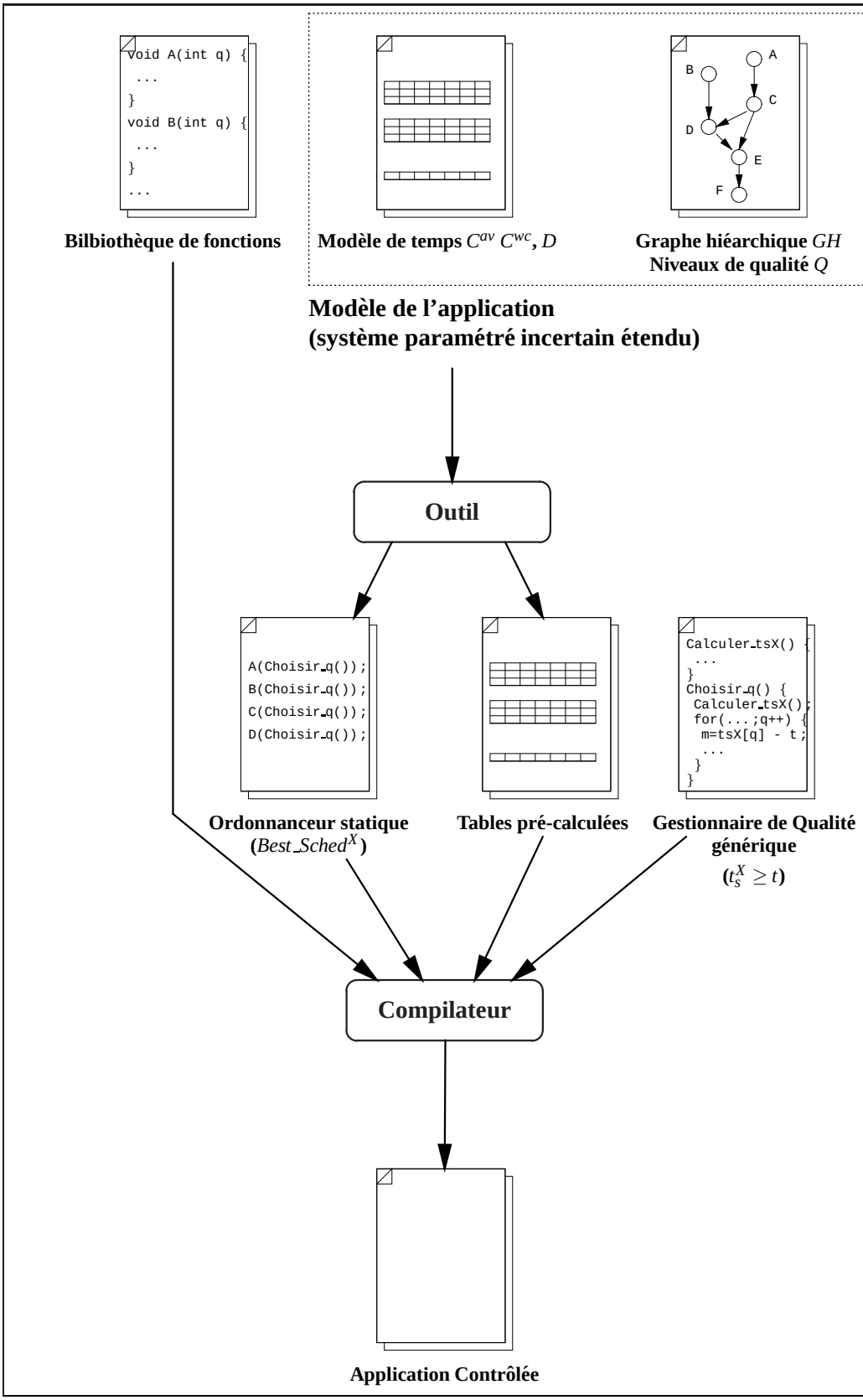


FIG. 4.1: Flot actuel de génération de l'application contrôlée.

Extension du modèle du graphe de flot de donnée aux boucles

4.1.2 Génération du contrôleur

L'outil qui a été développé s'intègre dans le flot de compilation et génération de l'application contrôlée. Il génère le contrôleur à partir du modèle de l'application. En pratique, deux fichiers sont générés :

Ordonnancement statique. Un fichier en langage C est généré par l'outil et est de la forme $\alpha = \{\alpha_1\}^{n_1}\{\alpha_2\}^{n_2}\ldots\{\alpha_m\}^{n_m}$, où les valeurs n_i ne sont pas connues statiquement, mais sont des paramètres d'entrée de l'application contrôlée. Dans la suite, nous appelons ***point de contrôle*** toute position dans le code de l'application contrôlée sur laquelle le contrôleur peut être appelé. Ces positions correspondent aux états du système paramétré incertain. Puisqu'il s'agit d'une politique d'ordonnancement statique, il existe également une correspondance directe entre les suffixes de $\alpha = \{\alpha_1\}^{n_1}\{\alpha_2\}^{n_2}\ldots\{\alpha_m\}^{n_m}$ et les points de contrôle.

Tables pré-calculées statiquement. Ces tables représentent l'implémentation du gestionnaire de qualité qui est spécifique à l'application. La partie générique, écrite directement en langage C, est donnée dans un fichier séparé. La génération des tables pré-calculées ainsi que l'implémentation du gestionnaire de qualité générique dépend de la politique de gestion de qualité choisie. Dans la suite, nous allons donner les algorithmes qui permettent la génération efficace des tables, ainsi que les algorithmes qui permettent calculer efficacement le niveau de qualité courant sur chaque point de contrôle.

4.1.3 Contrôleur

Le code C décrivant le fonctionnement du contrôleur est donné dans un fichier séparé. Ce code C est un code générique, et ne dépend pas de l'application à contrôler. Bien entendu, il utilise les valeurs pré-calculées qui sont générées par l'outil. Dans la suite, nous donnerons les algorithmes qui sont à la base de notre implémentation en C du contrôleur.

4.2 Graphe de précédence hiérarchique

Le modèle que nous traitons dans l'outil actuel est une extension de système paramétré incertain. Dans ce modèle étendu, nous prenons en compte la notion de *boucle*, c'est-à-dire la répétition d'un ensemble d'actions appelé *corps de boucle*. En effet, la plupart des applications multimédia comportent des traitements itératifs. En général, ces algorithmes sont implémentés à l'aide de boucles, comme dans le langage C par exemple. En introduisant une notion de boucle directement dans le modèle, nous facilitons d'une part la modélisation de l'application, et nous améliorons d'autre part l'efficacité du contrôleur, en évitant de déplier les boucles. Ainsi, la seule différence entre le modèle de système paramétré incertain utilisé dans les chapitres 2 et 3, et celui qui est utilisé en entrée de l'outil est l'utilisation d'un graphe de précédence hiérarchique à la place du graphe de précédence classique. Un graphe précédence hiérarchique permet de structurer un graphe de précédence en un

ensemble de boucles, dont les corps de boucle sont des graphes de précédence classique. Le nombre d'itérations de chaque boucle est alors un paramètre du modèle.

DÉFINITION 4.1 *Nous appelons* **graphe de précédence hiérarchique** *un graphe de précédence* $GH = (GS, \prec, \nu)$ *tel que :*

1. *GS est un ensemble fini de graphes de précédence,* $GS = \{ G_1, \ldots, G_N \}$ *avec* $G_i = (A_i, \prec_i)$*, tel que les ensembles d'actions* $A_1, \ldots, A_n$ *soient disjoints, c'est-à-dire pour tout* $i \neq j$ *nous avons* $A_i \cap A_j = \emptyset$*.*
2. $(GS, \prec)$ *est un graphe de précédence.*
3. ν *est un paramètre du type* $GS \rightarrow \mathbb{N}$*, qui fixe le nombre d'itérations des graphes de précédence* $G_1, \ldots, G_N$ *du graphe hiérarchique.*

A partir d'un graphe de précédence hiérarchique et du nombre d'itérations des boucles, il est possible de construire le graphe de précédence déplié, dans lequel les itérations des différentes actions sont représentées. Ce graphe est appelé *sémantique* du graphe hiérarchique. Il décrit les relations de précédence qui existent entre les différentes itérations des actions des différentes boucles qui composent le graphe hiérarchique. Il existe une dépendance entre deux itérations de deux actions dans les trois cas suivants :

- les deux actions sont dans deux boucles différentes et il existent un arc entre ces deux boucles dans le graphe hiérarchique ;
- il s'agit de la même itération de deux actions différentes d'une même boucle, et il existe un arc entre ces deux actions dans le graphe hiérarchique ;
- il s'agit de deux itérations différentes d'une même action, telles que le niveau de la première itération soit inférieur à celui de la seconde.

DÉFINITION 4.2 (sémantique d'un graphe hiérarchique) *Soient* $GH = (GS, \prec, \nu)$ *un graphe hiérarchique tel que* $GS = \{ G_1, \ldots, G_N \}$ *et pour tout* $i \in \{ 1, \ldots, N \}$, $G_i = (A_i, \prec_i)$*. Nous appelons alors* **sémantique** *du graphe hiérarchique GH par rapport au paramètre* ν*, le graphe de précédence noté* $||GH|| = (A_S, \prec_S)$ *tel que :*

1. *L'ensemble d'actions A est donné par :*

$$A_S = \bigcup_{1 \leq i \leq N} \Big(\bigcup_{1 \leq j \leq \nu(G_i)} \{ (a, j) \mid a \in A_i \} \Big).$$

2. *La relation de précédence* $\prec_S$ *est donnée par :*

$$(a,i) \prec_S (b,j) \in \prec \Leftrightarrow \begin{cases} a \in A_k \wedge b \in A_l \wedge G_k \prec G_l \\ a \in A_l \wedge b \in A_l \wedge a \prec_l b \\ a = b \wedge i < j. \end{cases}$$

La sémantique d'un graphe hiérarchique $GH = (GS, \prec, \nu)$ dépend du nombre d'itérations des boucles qui le composent, c'est-à-dire du paramètre ν. Dans la suite, nous cherchons à développer des

techniques de contrôle qui évitent de construire explicitement la sémantique du graphe hiérarchique, laissant le nombre d'itérations ν en paramètre. Afin de d'alléger les notations nous noterons a_i pour la $i^{\text{ème}}$ itération (a,i) d'une action a.

EXEMPLE 4.1 *Soit $GH = (GS, \prec, \nu)$ le graphe hiérarchique tel que $GS = \{ G_1, G_2 \}$, $G_i = (A_i, \prec_i)$ pour $i \in \{1,2\}$, $A_1 = \{a,b,c\}$ et $A_1 = \{d,f\}$. Les relations de précédence $\prec$, $\prec_1$ et $\prec_2$ sont représentées dans la figure 4.2. Dans cette figure, la hiérarchie est représentée par des rectangles. Ainsi, les actions a, b, c appartiennent au même rectangle.*

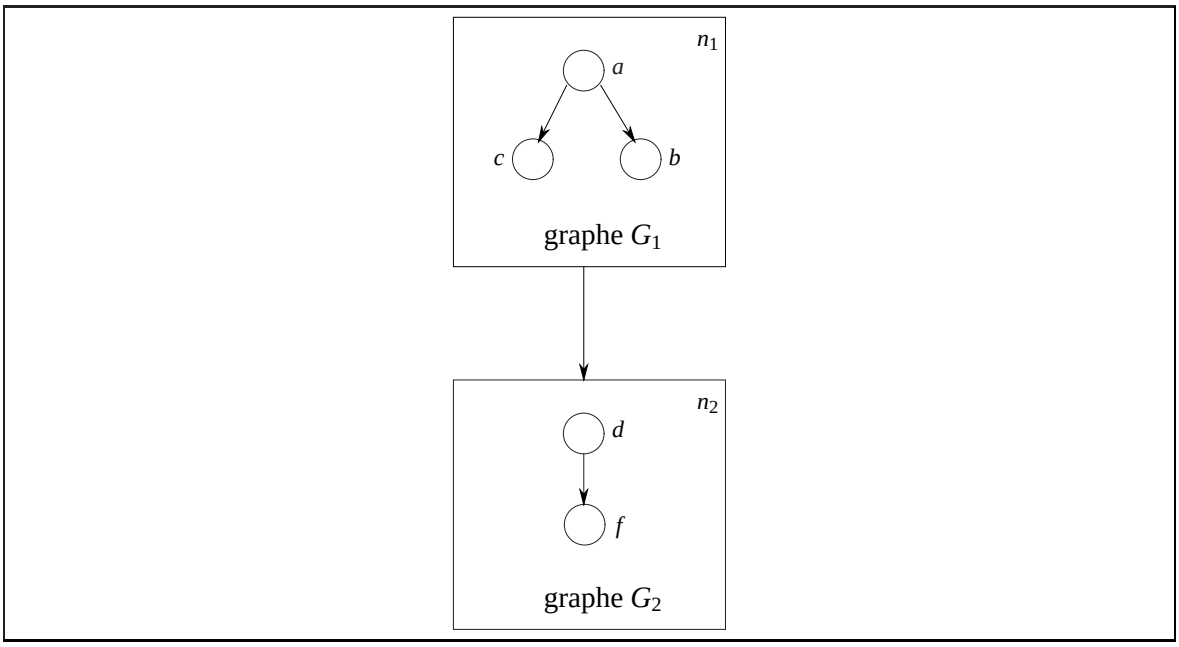


FIG. 4.2: Exemple de graphe hiérarchique GH.

La sémantique $||GH||$ du graphe GH pour l'affectation de nombre d'itérations $\nu : GS \to \mathbb{N}$, $\nu(G_1) = \nu_1 = 3$, $\nu(G_2) = \nu_2 = 2$, est représentée dans la figure 4.3. Nous rappelons que les représentations graphiques des graphes de précédence font apparaître seulement les arcs de la relation prédécesseur-successeur immédiats, afin de ne pas surcharger la représentation (voir la section 2.1.1).

Dans la suite, nous travaillerons avec des graphes de précédence hiérarchique à la place des graphes de précédence. Cela ne change pas fondamentalement le problème de contrôle d'un système paramétré incertain puisque la sémantique d'un graphe hiérarchique est un graphe de précédence. Lorsque nous parlerons de trace, prolongement ou ordonnancement d'un graphe hiérarchique, nous ferons en fait référence à la sémantique du graphe hiérarchique. Puisque les itérations d'une même action sont totalement ordonnées dans la sémantique du graphe hiérarchique, nous ne ferons pas apparaître les indices correspondant aux différentes itérations. Revenons sur le graphe hiérarchique GH de l'exemple 4.1 (figure 4.2). Si ν donne le nombre d'itérations des boucles de GH, et est telle que

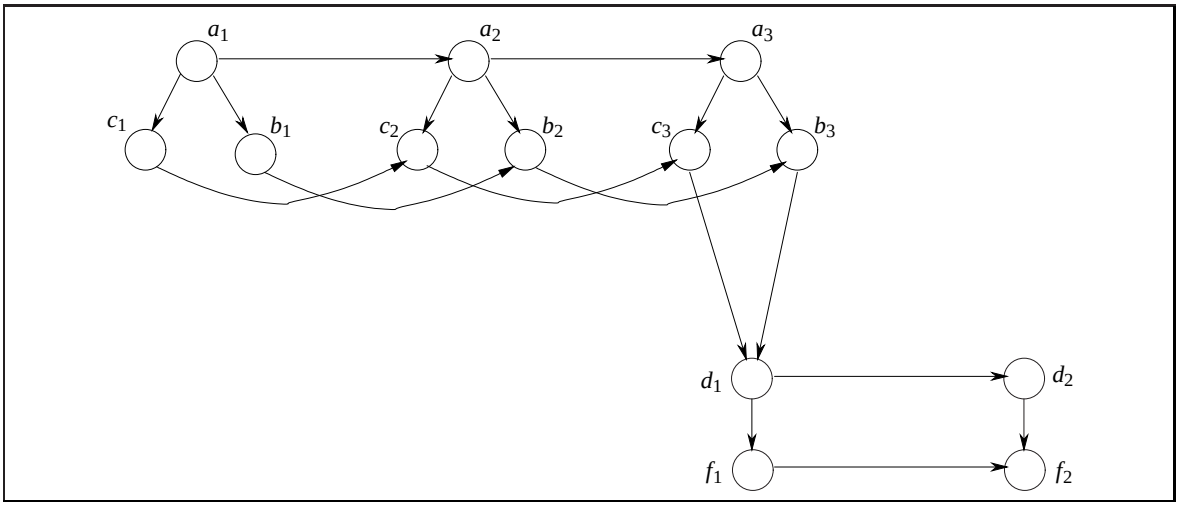


FIG. 4.3: Sémantique $||GH||$ du graphe GH pour l'affectation du nombre d'itérations $\nu_1 = 3$ et $\nu_2 = 2$.

$\nu(G_1) = 3$, $\nu(G_2) = 2$, alors nous dirons que $\alpha = abcabcabcddff$ est un ordonnancement de GH, puisqu'il correspond à l'ordonnancement $\alpha' = a_1b_1c_1a_2b_2c_2a_3b_3c_3d_1d_2f_1f_2$ de la sémantique $||GH||$ de GH.

Les techniques de génération de contrôleur que nous allons développer seront indépendantes du nombre d'itération des boucles. Ceci permet de générer une application contrôlée générique, paramétrée par le nombre d'itérations connu à l'exécution seulement.

4.3 Ordonnancement d'un graphe hiérarchique

Nous expliquons ici comment générer une politique d'ordonnancement pour un système paramétré incertain dont le graphe de précédence est hiérarchique. Nous cherchons une technique qui évite le dépliage du graphe hiárchique, c'est-à-dire qui ne travaille pas sur sa sémantique mais sur le graphe hiérarchique lui-même. Ceci permet de laisser le nombre d'itérations des boucles comme un paramètre d'entrée de l'application. De plus, la technique est plus efficace qu'une technique basée sur un dépliage, surtout lorsque le nombre d'itérations des boucles est grand, ce qui est le cas de l'application considérée dans le chapitre 5. L'opérateur de répétition suivant est utilisé pour décrire des ordonnancements dans lesquels le nombre de répétitions d'un sous-ensemble d'actions peut ne pas être statiquement connu et dépendre du nombre d'itérations des boucles du graphe hiérarchique.

NOTATION 4.1 *Soit A un ensemble d'actions et α une séquence d'actions. Nous noterons $\{\alpha\}^\nu$ la séquence de longueur $\nu|\alpha|$ correspondant à la répétition de la séquence α ν fois, c'est-à-dire :*

$$\{\alpha\}^\nu = \{\alpha\}^\nu = \underbrace{\alpha\alpha\ldots\alpha}_{\nu \text{ fois}}.$$

Etant donnée une boucle $G_i = (A_i, \prec_i)$, il est possible d'ordonnancer une seule de ses itérations, c'est-à-dire ordonnancer le graphe G_i. Soit α_i un ordonnancement du graphe G_i. En répétant cet l'ordonnancement α_i autant de fois que le nombre d'itérations de la boucle G_i, c'est-à-dire $\nu(G_i)$, nous obtenons l'ordonnancement $\{\alpha_i\}^{\nu(G_i)}$ de la boucle G_i. Nous obtenons de la sorte un ordonnancement dont la représentation est très compacte, ce qui n'est pas le cas de tous les ordonnancements possibles de cette boucle. Cependant, pour pouvoir optimiser l'ordonnancement d'une boucle par rapport à une politique de gestion de qualité donnée, il peut être nécessaire de ne pas se limiter aux ordonnancement de la forme $\{\alpha_i\}^{\nu(G_i)}$. Dans ce cas, il convient de trouver un compromis entre la compacité de l'ordonnancement, et son optimalité par rapport à la politique de gestion de qualité choisie.

Soit $GH = (GS, \prec, \nu)$ un graphe hiérarchique. La forme générale d'un ordonnancement de GH sera $\{\alpha_1\}^{\nu_1}\{\alpha_2\}^{\nu_2}\ldots\{\alpha_m\}^{\nu_m}$, où $\nu_1, \ldots, \nu_m$ sont des expressions qui peuvent dépendre du paramètre ν. Dans la suite, nous dirons que la séquence $\{\alpha_i\}^{\nu_i}$ est une ***boucle***, même si elle ne correspond pas à l'ordonnancement d'une boucle du graphe hiérarchique. Nous dirons de plus que α_i est le ***corps de la boucle*** $\{\alpha_i\}^{\nu_i}$, puisqu'il s'agit du motif de base qui est répété dans la boucle $\{\alpha_i\}^{\nu_i}$.

Politique d'ordonnancement statique EDF

La génération d'un ordonnancement EDF d'un graphe hiérarchique $GH = (GS, \prec, \nu)$, $GS = \{ G_1, \ldots, G_N \}$, peut être fait de la manière suivante. Soit $D : A_1 \cup \ldots \cup A_N \rightarrow \mathbb{R}^+$ une fonction d'échéance. Supposons que la fonction d'échéance D est constante sur chaque boucle, c'est-à-dire D est constante sur chaque sous-ensemble d'actions A_i. Si ce n'est pas le cas, nous transformons le graphe hiérarchique en découpant chaque boucle G_i de GH selon la partition induite par la fonction D_i^* des échéances rétro-propagées dans G_i, ce qui transforme la boucle G_i en autant de boucles que la taille de la partition. Les arcs de précédence entre les boucles ainsi obtenus sont placés de façon à respecter la règle EDF et les précédence initiaux. Ce principe est illustré par la figure 4.4, qui reprend le graphe hiérarchique de l'exemple 4.1. Les valeurs prises par la fonction d'échéance D ont été ajoutées, et nous illustrons le découpage de la boucle G_1 sur laquelle D n'est pas constante. Après transformation, les valeurs correspondent aux échéances rétro-propagées.

Puisque la fonction D est constante sur les boucles G_i de GH, le graphe GH peut être vu comme un graphe de précédence classique. Il est donc possible de construire l'ordonnancement EDF $\gamma = \gamma(1)\ldots\gamma(N)$ de GH vu comme un graphe de précédence, et par rapport à la fonction d'échéance D. Les éléments $\gamma(1)$, ..., $\gamma(N)$ de γ sont donc les boucles G_1, ..., G_N de GH. L'ordonnancement final α du graphe hiérarchique GH est obtenu en concaténant les ordonnancements α_i des boucles G_i de GH, dans l'ordre donné par α, c'est-à-dire $\alpha = \{\alpha_1\}^{\nu_1}\ldots\{\alpha_N\}^{\nu_N}$, où α_i est un ordonnancement de la boucle $\gamma(i)$.

4.4 Mise en œuvre efficace des politiques de gestion de qualité

Nous allons voir comment implémenter les politiques de gestion de qualité qui ont été proposées dans le chapitre 3, c'est-à-dire la politique de gestion de qualité simple et la politique de gestion de qualité mixte. Concernant la politique de gestion de qualité simple, nous introduisons d'abord la

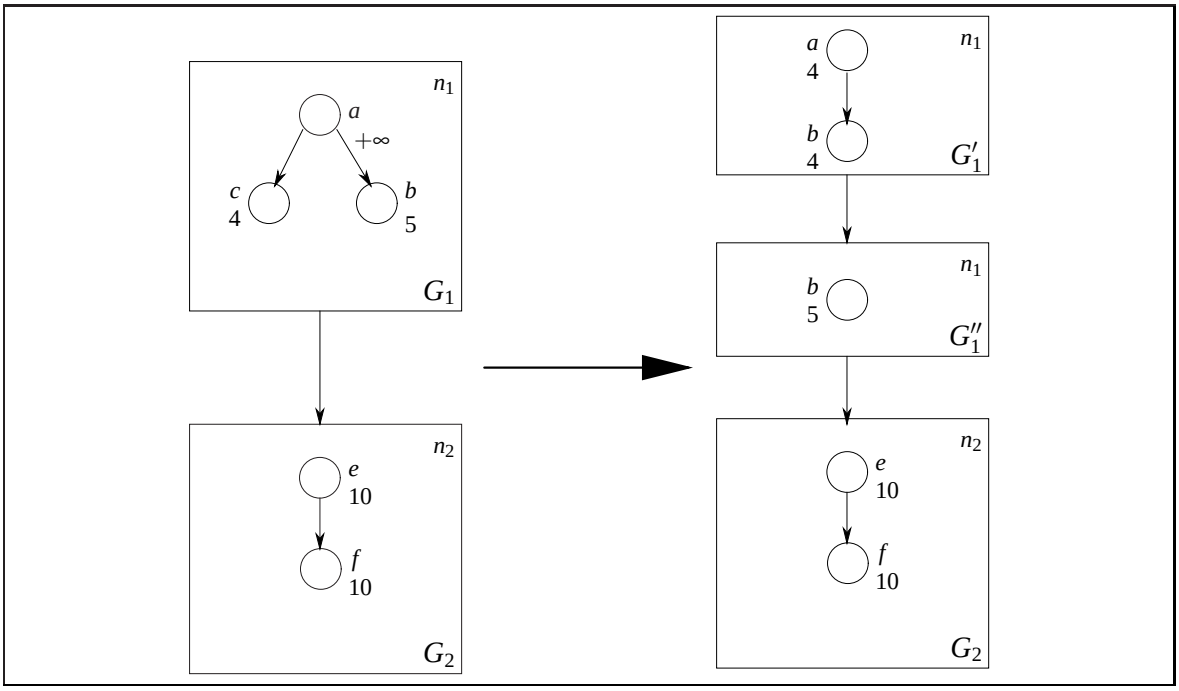


FIG. 4.4: Transformation du graphe hiérarchique.

notion de fonction de durée d'exécution *à mise-à-jour rapide*. Nous allons voir comment implémenter une politique de gestion de qualité basée sur une fonction de durée d'exécution *à mise-à-jour rapide*.

4.4.1 Mise en œuvre efficace de la politique de gestion de qualité simple

Dans cette section, nous introduisons la notion de fonction de durée d'exécution *à mise-à-jour rapide*. La fonction d'ordonnancement associée à une fonction de durée d'exécution à mise-à-jour rapide possède un certain nombre de propriétés qui permettent calculer facilement la valeur de la fonction d'ordonnancement sur tous les points de contrôle, dans le cas de l'utilisation d'une politique d'ordonnancement statique. En effet, dans ce cas la valeur de la fonction d'ordonnancement sur un point de contrôle est déduite de la valeur au point de contrôle précédent.

Notion de fonction de durée d'exécution à mise-à-jour rapide

Une fonction de durée d'exécution $C^X : A^* \times \Theta \rightarrow \mathbb{R}^+$ à mise-à-jour rapide est telle que la différence entre la durée d'une séquence d'action α et un de ses suffixes α^k, c'est-à-dire $C^X(\alpha,\theta) - C^X(\alpha^k,\theta)$, est indépendante des actions $\alpha(k+1)$, ..., $\alpha(|\alpha|)$.

DÉFINITION 4.3 *Soit $C^X : A^* \times \Theta \rightarrow \mathbb{R}^+$ une fonction de durée d'exécution. Nous dirons que C^X est une fonction de durée d'exécution* **à mise-à-jour rapide** *si pour toutes séquences α_1 et α_2, toute*

action a et toute affectation de qualité θ *nous avons :*

$$C^X(\alpha_1 a,\theta) - C^X(a,\theta) = C^X(\alpha_1 a\alpha_2,\theta) - C^X(a\alpha_2,\theta).$$

PROPOSITION 4.1 *Les fonctions de durée d'exécution* C^{av} *et* C^{sf} *sont des fonctions de durée d'exécution à mise-à-jour rapide. Plus précisément :*

$$\begin{aligned} C^{av}(\alpha a,\theta) - C^{av}(a,\theta) &= C^{av}(\alpha,\theta) \\ C^{sf}(\alpha a,\theta) - C^{sf}(a,\theta) &= C^{sf}(\alpha,\theta) + C^{wc}(a,q_{min}) - C^{wc}(a,\theta). \end{aligned}$$

Preuve de la proposition : Puisque la fonction $C^{av} : A^* \times \Theta \to \mathbb{R}^+$ est obtenue par extension de la fonction $C^{av} : A \times Q \to \mathbb{R}^+$, nous avons :

$$\begin{aligned} C^{av}(\alpha_1 a\alpha_2,\theta) - C^{av}(a\alpha_2,\theta) &= C^{av}(\alpha_1 a,\theta) + C^{av}(\alpha_2,\theta) - (C^{av}(a,\theta) + C^{av}(\alpha_2,\theta)) \\ &= C^{av}(\alpha_1 a,\theta) - C^{av}(a,\theta) = C^{av}(\alpha_1,\theta). \end{aligned}$$

En utilisant la définition de la fonction C^{sf}, nous avons :

$$\begin{aligned} C^{sf}(\alpha_1 a\alpha_2,\theta) - C^{sf}(a\alpha_2,\theta) &= C^{wc}(\alpha_1(1),\theta) + C^{wc}({\alpha_1}^2 a, q_{min}) + C^{wc}(\alpha_2, q_{min}) \\ &\quad -(C^{wc}(a,\theta) + C^{wc}(\alpha_2, q_{min})) \\ &= C^{wc}(\alpha_1(1),\theta) + C^{wc}({\alpha_1}^2, q_{min}) + C^{wc}(a, q_{min}) - C^{wc}(a,\theta) \\ &= C^{sf}(\alpha_1,\theta) + C^{wc}(a, q_{min}) - C^{wc}(a,\theta). \quad \square \end{aligned}$$

Les fonctions de durée d'exécution C^{av} et C^{sf} sont des fonctions de durée d'exécution à mise-à-jour rapide. Ainsi, les résultats que nous allons voir dans cette section seront applicables dans le cas de la politique de gestion de qualité moyenne et sûre. Lorsque nous le jugerons nécessaire, nous instancierons ces résultats généraux à l'une ou l'autre des politiques.

PROPOSITION 4.2 (calcul incrémental de t_s^X) *Soit* C^X *une fonction de durée d'exécution à mise-à-jour rapide,* (α,θ) *un prolongement et* j *l'indice de l'échéance critique de* (α,θ) *par rapport à* t_s^X*. Alors, pour tout* $i \leq j$ *nous avons :*

$$t_s^X(\alpha^i,\theta) = t_s^X(\alpha,\theta) + C^X({}^i\alpha,\theta) - C^X(\alpha(i),\theta).$$

Preuve de la proposition : Tout d'abord, nous exprimons les valeurs $t_s^X(\alpha^i,\theta)(k)$ en fonction des valeurs $t_s^X(\alpha,\theta)(k-i+1)$:

$$\begin{aligned} t_s^X(\alpha^i,\theta)(k) &= D({}^k(\alpha^i)) - C^X({}^k(\alpha^i),\theta) \\ &= D(\alpha[i,k-i+1]) - C^X(\alpha[i,k-i+1],\theta) \\ &= D(\alpha[i,k-i+1]) - C^X(\alpha[i,k-i+1],\theta) + C^X(\alpha[1,k-i+1],\theta) - C^X(\alpha[1,k-i+1],\theta). \end{aligned}$$

Puisque C^X est une fonction de durée d'exécution à mise-à-jour rapide, nous obtenons :

$$\begin{aligned} t_s^X(\alpha^i,\theta)(k) &= D(\alpha[i,k-i+1]) - C^X(\alpha[1,k-i+1],\theta) + C^X(\alpha[1,i],\theta) - C^X(\alpha(i),\theta) \\ &= t_s^X(\alpha,\theta)(k-i+1) + C^X({}^i\alpha,\theta) - C^X(\alpha(i),\theta). \end{aligned}$$

Le résultat précédent nous permet d'écrire :

$$\begin{aligned} t_s^X(\alpha^i,\theta) &= \mathbf{max}_{1\leq k\leq|\alpha^i|}\, t_s^X(\alpha^i,\theta)(k) \\ &= \mathbf{max}_{1\leq k\leq|\alpha^i|}\, t_s^X(\alpha,\theta)(k-i+1) + C^X({}^i\alpha,\theta) - C^X(\alpha(i),\theta) \\ &= C^X({}^i\alpha,\theta) - C^X(\alpha(i),\theta) + \mathbf{max}_{1\leq k\leq|\alpha^i|}\, t_s^X(\alpha,\theta)(k-i+1) \end{aligned}$$

Puisque le maximum des valeurs $t_s^X(\alpha,\theta)(k-i+1)$ est atteint lorsque $j = k-i+1$, c'est-à-dire lorsque $k = j-i+1 \in \{1,\ldots,|\alpha^i|\}$, nous avons :

$$\begin{aligned} &= C^X({}^i\alpha,\theta) - C^X(\alpha(i),\theta) + \mathbf{max}_{1\leq k\leq|\alpha|}\, t_s^X(\alpha,\theta)(k) \\ &= C^X({}^i\alpha,\theta) - C^X(\alpha(i),\theta) + t_s^X(\alpha,\theta). \quad \square \end{aligned}$$

La proposition précédente permet de déduire la valeur de la fonction d'ordonnancement t_s^X d'un point de contrôle à un autre. En particulier, entre deux points de contrôle consécutifs, c'est-à-dire pour $i = 1$, et pour une affectation de qualité constante, c'est-à-dire $\theta = q$, nous avons :

$$t_s^X(\alpha^2,q) = t_s^X(\alpha,q) + C^X({}^2\alpha,q) - C^X(\alpha(2),q).$$

Ainsi, si le contrôleur ne remet pas en cause l'ordonnancement prévu, et en particulier si la politique d'ordonnancement est statique, il est possible de déduire la valeur de la fonction d'ordonnancement d'un point de contrôle à un autre par un calcul très simple.

PROPOSITION 4.3 (calcul incrémental de C^X) *Soit C^X une fonction de durée d'exécution à mise-à-jour rapide. Soient α_1 et α_2 deux séquences d'actions, et θ une affectation de qualité définie sur $\mathsf{ens}(\alpha_1\alpha_2)$. Nous notons a la dernière action de α_1, c'est-à-dire $a = \alpha_1(|\alpha_1|)$, et b la première action de α_2, c'est-à-dire $b = \alpha_2(1)$. Alors nous avons :*

$$C^X(\alpha_1\alpha_2,\theta) = C^X(\alpha_1,\theta) + C^X(\alpha_2,\theta) + C^X(ab,\theta) - C^X(a,\theta) - C^X(b,\theta).$$

Preuve de la proposition : Nous notons α_1' le préfixe de α_1 tel que $\alpha_1' a = \alpha_1$, c'est-à-dire $\alpha_1' = {}^{|\alpha_1|-1}\alpha_1$, et α_2' le suffixe de α_2 tel que $b\alpha_2' = \alpha_2$, c'est-à-dire $\alpha_2' = {\alpha_2}^2$. Nous avons alors :

$$C^X(\alpha_1\alpha_2,\theta) = C^X(\alpha_1 b\alpha_2',\theta).$$

Puisque C^X est une fonction de durée d'exécution à mise-à-jour rapide, nous avons $C^X(\alpha_1 b\alpha_2',\theta) - C^X(b\alpha_2',\theta) = C^X(\alpha_1 b,\theta) - C^X(b,\theta)$, c'est-à-dire $C^X(\alpha_1 b\alpha_2',\theta) = C^X(\alpha_1 b,\theta) -$

$C^X(b,\theta)+C^X(b\alpha'_2,\theta)$. Ainsi, nous obtenons :

$$\begin{aligned} C^X(\alpha_1\alpha_2,\theta) &= C^X(\alpha_1 b,\theta)-C^X(b,\theta)+C^X(b\alpha'_2,\theta) \\ &= C^X(\alpha'_1 ab,\theta)-C^X(b,\theta)+C^X(\alpha_2,\theta) \\ &= C^X(\alpha'_1 a,\theta)-C^X(a,\theta)+C^X(ab,\theta)-C^X(b,\theta)+C^X(\alpha_2,\theta) \\ &= C^X(\alpha_1,\theta)-C^X(a,\theta)+C^X(ab,\theta)-C^X(b,\theta)+C^X(\alpha_2,\theta). \quad \square \end{aligned}$$

COROLLAIRE 4.1 *Soit $\alpha = \{\alpha_1\}^{n_1}\{\alpha_2\}^{n_2}\ldots\{\alpha_m\}^{n_m}$ un ordonnancement et C^X une fonction de durée d'exécution à mise-à-jour rapide. Nous notons a_k et b_k les actions qui correspondent aux premières et dernières actions des ordonnancements α_k, c'est-à-dire $a_k = \alpha_k(1)$ et $b_k = \alpha_k(|\alpha_k|)$. Nous avons :*

$$C^X(\{\alpha_k\}^{n_k},\theta) = n_k \cdot C^X(\alpha_k,\theta) + (n_k-1)\cdot(C^X(b_k a_k,\theta)-C^X(b_k,\theta)-C^X(a_k,\theta)).$$

Nous en déduisons l'expression de $C^X(\alpha,\theta)$ suivante :

$$\begin{aligned} C^X(\alpha,\theta) = & \sum_{1\le k\le m} n_k\cdot C^X(\alpha_k,\theta)+(n_k-1)\cdot(C^X(b_k a_k,\theta)-C^X(b_k,\theta)-C^X(a_k,\theta)) \\ & + \sum_{1\le k\le m-1}(C^X(b_k a_{k+1},\theta)-C^X(b_k,\theta)-C^X(a_k,\theta)). \end{aligned}$$

Preuve du corollaire :
La preuve est faite par récurrence sur n_i. Soit $\mathcal{P}(n)$ la proposition suivante :

$$\mathcal{P}(n) \Leftrightarrow C^X(\{\alpha_k\}^n,\theta) = n\cdot C^X(\alpha_k,\theta)+(n-1)\cdot(C^X(b_k a_k,\theta)-C^X(b_k,\theta)-C^X(a_k,\theta)).$$

Nous démontrons d'abord que la proposition $\mathcal{P}(1)$ est vraie. En effet, nous avons $\mathcal{P}(1) \Leftrightarrow C^X(\{\alpha_k\}^1,\theta) = C^X(\alpha_k,\theta)$.

Nous montrons à présent que $\mathcal{P}(n) \Rightarrow \mathcal{P}(n+1)$. Supposons que la proposition $\mathcal{P}(n)$ est vraie. D'après la proposition 4.3, nous avons :

$$C^X(\{\alpha_k\}^{n+1},\theta) = C^X(\{\alpha_k\}^n\alpha_k,\theta) = C^X(\{\alpha_k\}^n,\theta)+C^X(\alpha_k,\theta)+C^X(b_k a_k,\theta)-C^X(b_k,\theta)-C^X(a_k,\theta).$$

Puisque nous supposons que $\mathcal{P}(n)$ est vraie, nous obtenons :

$$\begin{aligned} C^X(\{\alpha_k\}^{n+1},\theta) &= C^X(\{\alpha_k\}^n,\theta)+C^X(\alpha_k,\theta)+C^X(b_k a_k,\theta)-C^X(b_k,\theta)-C^X(a_k,\theta) \\ &= n\cdot C^X(\alpha_k,\theta)+(n-1)\cdot(C^X(b_k a_k,\theta)-C^X(b_k,\theta)-C^X(a_k,\theta)) \\ &\quad +C^X(\alpha_k,\theta)+C^X(b_k a_k,\theta)-C^X(b_k,\theta)-C^X(a_k,\theta) \\ &= (n+1)\cdot C^X(\alpha_k,\theta)+n\cdot(C^X(b_k a_k,\theta)-C^X(b_k,\theta)-C^X(a_k,\theta)). \end{aligned}$$

Ainsi, nous avons démontré $\mathcal{P}(n) \Rightarrow \mathcal{P}(n+1)$, ce qui termine la démonstration par récurrence du résultat.

L'expression de $C^X(\alpha,\theta)$ se déduit du résultat précédent en appliquant une nouvelle fois le résultat de composition de la proposition 4.3 :

$$\begin{aligned} C^X(\alpha,\theta) &= C^X(\{\alpha_1\}^{n_1}\{\alpha_2\}^{n_2}\ldots\{\alpha_m\}^{n_m},\theta) \\ &= C^X(\{\alpha_1\}^{n_1},\theta)+\ldots+C^X(\{\alpha_k\}^{n_k},\theta)+C^X(b_1a_2,\theta)+\ldots+C^X(b_{m-1}a_m,\theta) \\ &\quad -C^X(b_1,\theta)-\ldots-C^X(b_{m-1},\theta)-C^X(a_2,\theta)-\ldots C^X(a_m,\theta). \end{aligned}$$

En remplaçant $C^X(\{\alpha_k\}^{n_k},\theta)$ par l'expression trouvé dans la première partie du corollaire, nous obtenons le résultat. □

PROPOSITION 4.4 (invariance de l'échéance critique) *Soient C^X une fonction de durée d'exécution à mise-à-jour rapide et (α,θ) un prolongement quelconque d'un état d'un système paramétré incertain $SPI(C)$. Considérons l'indice j de l'échéance critique de (α,θ) par rapport à C^X. Alors, pour tout $i \leq j$, l'indice de l'échéance critique de (α^i,θ) par rapport à C^X est $j-i+1$.*

Preuve de la proposition : Pour tout k nous avons :

$$t_s^X(\alpha^i,\theta)(k) = t_s^X(\alpha,\theta)(k+i-1)+C^X({}^i\alpha,\theta)-C^X(\alpha(i),\theta).$$

Nous en déduisons :

$$\begin{aligned} \{\, k\geq 1 \mid t_s^X(\alpha^i,\theta)(k)=t_s^X(\alpha^i,\theta)\,\} &= \{\, k\geq 1 \mid t_s^X(\alpha,\theta)(k+i-1)=t_s^X(\alpha,\theta)\,\} \\ \Rightarrow \mathbf{max}\{\, k\geq 1 \mid t_s^X(\alpha^i,\theta)(k)=t_s^X(\alpha^i,\theta)\} &= \mathbf{max}\{\, k\geq 1 \mid t_s^X(\alpha,\theta)(k+i-1)=t_s^X(\alpha,\theta)\} \\ \Rightarrow \mathbf{max}\{\, k\geq 1 \mid t_s^X(\alpha^i,\theta)(k)=t_s^X(\alpha^i,\theta)\} &= \mathbf{max}\{\, k\geq i \mid t_s^X(\alpha,\theta)(k)=t_s^X(\alpha,\theta)\}-i+1. \end{aligned}$$

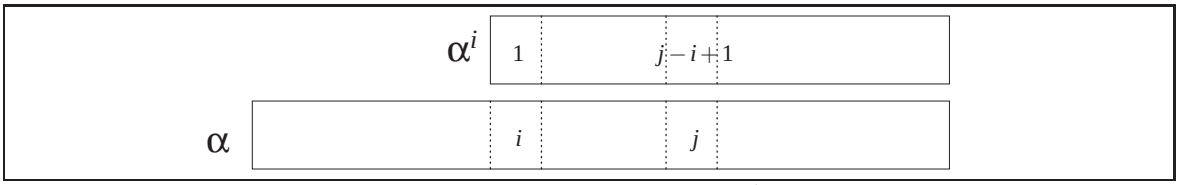


FIG. 4.5: Prolongements α et α^i.

Puisque nous avons l'indice de l'échéance critique de (α,θ) est j et $i \leq j$, nous avons :

$$j = \mathbf{max}\{\, k\geq 1 \mid t_s^X(\alpha,\theta)(k)=t_s^X(\alpha,\theta)\} = \mathbf{max}\{\, k\geq i \mid t_s^X(\alpha,\theta)(k)=t_s^X(\alpha,\theta)\}.$$

Nous pouvons donc conclure que :

$$\mathbf{max}\{\, k\geq 1 \mid t_s^X(\alpha^i,\theta)(k)=t_s^X(\alpha^i,\theta)\} = j-i+1,$$

c'est-à-dire que l'indice de l'échéance critique de (α^i,θ) est $j-i+1$. □

La proposition précédente donne un résultat très intéressant à propos des fonctions de durée d'exécution à mise-à-jour rapides. En effet, l'indice j de l'échéance critique d'un prolongement (α,θ)

devient $j-i+1$ après avoir exécuté les $i-1$, $i \leq j$, premières actions. Autrement dit, dans le cas d'une politique d'ordonnancement statique, l'échéance critique ne change tant que l'action qui correspond à cette échéance n'a pas été exécutée.

Fort de ces résultats sur les fonctions de durée d'exécution à mise-à-jour rapide, nous proposons dans ce qui suit une implémentation efficace du contrôleur Γ^X lorsque la politique d'ordonnancement est statique, et que la fonction de durée d'exécution C^X est à mise-à-jour rapide.

4.4.2 Implémentation efficace dans le cas d'un ordonnanceur statique

Nous allons voir comment implémenter efficacement un contrôleur basé sur une fonction de durée d'exécution à mise-à-jour rapide et ne politique d'ordonnancement statique. Dans un premier temps nous expliquons comment pré-calculer un certain nombre de valeurs nécessaires, utilisées pour accélérer l'exécution du contrôleur. Nous donnons ensuite l'algorithme de calcul du contrôleur en tout point ce contrôle. Il s'agit d'une instanciation efficace de l'algorithme de contrôle abstrait dans ce contexte particulier. Les optimisations sont basées sur les résultats de la section précédente, et en particulier sur ceux du corollaire 4.1 et de la proposition 4.4.

Soit $SPI(C) = (GH, C^{wc}, D, Q; C)$ un système paramétré incertain dont le graphe de précédence $GH = (GS, \prec, \nu)$ est hiérarchique. Soit C^X une fonction de durée d'exécution à mise-à-jour rapide et une politique d'ordonnancement statique basé sur l'ordonnancement $\alpha = \{\alpha_1\}^{n_1}\{\alpha_2\}^{n_2}\ldots\{\alpha_m\}^{n_m}$ du graphe hiérarchique GH. Nous notons α_0 l'ordonnancement des corps de boucle, c'est-à-dire $\alpha_0 = \alpha_1\alpha_2\ldots\alpha_m$.

Valeurs pré-calculées statiquement

Pour plus de clarté dans l'exposé, nous avons choisi de représenter les valeurs pré-calculées avec une police de caractère de machine à écrire. Tout d'abord, nous définissons la variable représentant le nombre total de boucles dans l'ordonnancement α :

$$\texttt{NOMBRE\_BOUCLES} := m.$$

Nous définissons ensuite les valeurs $\texttt{maj\_action}^X[i,q]$ pour tout $i \in \{\ 1,\ldots,|\alpha|-1\ \}$. Celles-ci permettent de mettre à jour la valeur de la fonction d'ordonnancement entre deux point de contrôle, lorsque l'échéance critique ne change pas.

$$\texttt{maj\_action}^X[i,q] := C^X(\alpha_0[i,i+1],q) - C^X(\alpha_0(i+1),q).$$

Pour finir, nous définissons les trois tables de mise à jour suivantes. Elles correspondent au niveau de granularité des boucles. Ces tables sont utiles lors de l'initialisation de la valeur de la fonction d'ordonnancement, ainsi que lors du "franchissement" d'une échéance critique, c'est-à-dire après l'exécution d'une action dont l'échéance correspond à l'échéance critique. Soient a_k et b_k les actions qui correspondent aux premières actions et aux dernières actions des corps de boucles α_k, c'est-à-dire $a_k = \alpha_k(1)$ et $b_k = \alpha_k(|\alpha_k|)$.

$$\texttt{maj\_echeance}^X[k,q] \quad := \quad C^X(\alpha_k,q)$$

NOMBRE_BOUCLES	nombres de boucles composant l'ordonnancement α
maj_action$^X[i,q]$	mise à jour de t_s^X entre deux actions
maj_echeance$^X[k,q]$	durées d'exécution des corps de boucle
maj_echeance_correction_intra$^X[k,q]$	calcul de la durée d'exécution des boucles
maj_echeance_correction_inter$^X[k,q]$	calcul de la durée d'exécution de l'ordonnancement

FIG. 4.6: Valeurs pré-calculées statiquement.

echeances$[k]$	bornes des boucles de l'ordonnancement
nombre_iterations$[k]$	valeur des échéances

FIG. 4.7: Paramètres connus à l'exécution.

$$\texttt{maj\_echeance\_correction\_intra}^X[k,q] := C^X(b_k a_k, q) - C^X(a_k, q) - C^X(b_k, q)$$

$$\texttt{maj\_echeance\_correction\_inter}^X[k,q] := \begin{cases} C^X(b_k a_{k+1}, q) - C^X(b_k, q) - C^X(a_{k+1}, q) \text{ si } k < m \\ 0 \text{ sinon.} \end{cases}$$

Algorithmes de calcul de Γ^X

Nous allons donner les algorithmes de calcul du contrôleur Γ^X basé sur la fonction de durée d'exécution à mise-à-jour rapide C^X et une politique d'ordonnancement statique. Les algorithmes présentés concernent essentiellement le calcul de la fonction t_s^X. Nous utilisons quatre tableaux globaux afin de mémoriser certaines informations entre les points de contrôle. Ces tableaux sont initialisés par la fonction *Initialiser_*t_s^X (figure 4.11). Ainsi, la valeur $t_s^X[q]$ représente la valeur de la fonction d'ordonnancement sur le point de contrôle courant au niveau de qualité q. Le tableau t_s^X*_pile* est une pile correspondant au valeurs prises par t_s^X lors des franchissement des échéances critiques. La tableau t_s^X*_pile_indice* donne l'indice des boucles correspondant aux échéances critiques. Pour finir, le tableau t_s^X*_pile_taille* donne la taille des deux piles précédentes pour les différents niveaux de qualité. Cette taille est bornée par le nombre de boucle de l'ordonnancement α, c'est-à-dire par m.

Le calcul de la fonction t_s^X utilise les valeurs statiquement pré-calculées, c'est-à-dire les tables NOMBRE_BOUCLES et les tables maj_action^X, maj_echeance^X, maj_echeance_correction_intra^X et maj_echeance_correction_inter^X. En outre, la valeur des échéances et le nombre d'itérations de chaque boucle est connu au démarrage de l'application. Pour se faire, nous supposons que les tables nombre_iterations et echeances

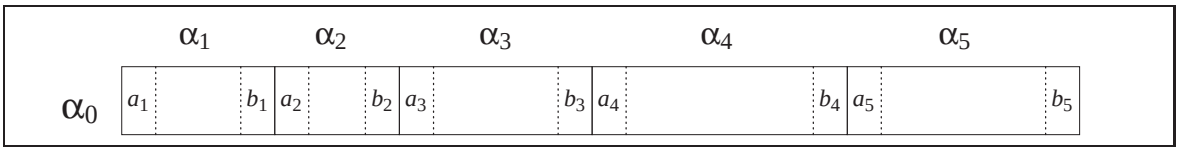


FIG. 4.8: Ordonnancement $\alpha_0 = \alpha_1 \alpha_2 \ldots \alpha_m$.

sont initialisées à l'exécution comme suit :

$$\begin{aligned} \texttt{nombre\_iterations}[k] &:= n_k \\ \texttt{echeance}[k] &:= D(\alpha_k). \end{aligned}$$

$t_s^X[q]$	contient la valeur courante t_s^X pour le niveau de qualité q
$t_s^X\_pile[i,q]$	piles des valeurs de t_s^X au changement d'échéance critique
$t_s^X\_pile\_indice[i,q]$	piles des indice des échéances critiques
$t_s^X\_pile\_taille[q]$	taille des piles $t_s^X\_pile$ et $t_s^X\_pile\_indice$

FIG. 4.9: Tableaux globaux utilisés pour le calcul de t_s^X.

A chaque point de contrôle, un appel à la fonction *Choisir_q_X* (figure 4.11) permet de calculer le niveau de qualité de la prochaine action à exécuter. Ainsi, la fonction *Choisir_q_X* prend un argument le triplet (i,k,t) et retourne un niveau de qualité. Le triplet (i,k,t) donne une information sur l'état courant : i correspond à l'indice de l'action qui vient d'être exécutée, plus précisément à une position dans l'ordonnancement α_0, k correspond à l'indice de la boucle courante, et t est le temps-réel.

La fonction *Choisir_q_X* fait appel à $Calculer\_t_s^X$. Cette dernière calcule, pour chaque niveau de qualité q, la valeur courante $t_s^X[q]$. Tant que l'échéance critique ne change pas, c'est-à-dire tant que $k < echeance\_critique^X$, le calcul $t_s^X[q]$ sur le point de contrôle courant suivant se fait par une simple correction par rapport à la valeur courante. Cette correction est réalisée par la fonction $MAJ\_Action\_t_s^X$. Lorsque l'échéance critique est dépassée, c'est-à-dire lorsque $k \geq echeance\_critique^X[q]$, la valeur $t_s^X[q]$ sur le point de contrôle suivant ne peut pas être déduite de la valeur courante. Dans ce cas, $t_s^X[q]$ est ré-initialisé avec la valeur qui se trouve au sommet de la pile $t_s^X\_pile$. C'est la fonction $MAJ\_Boucle\_t_s^X$ permet cette ré-initialisation.

```
Choisir_q_X(i,k,t)
    si (i = 0) faire Initialiser_t_s^X()
    Calculer_t_s^X(i,k)
    q_M := q_min
    pour tout q ∈ Q faire
        si (q ≥ q_M ∧ t_s^X[q] ≥ t) faire q_M := q
    fin pour tout
    retourner q_M
fin Choisir_q_X
```

FIG. 4.10: Algorithme de contrôle du niveau de qualité, c'est-à-dire implémentation d'algorithme de contrôle abstrait.

```
Initialiser_t_s^X()
  pour tout q ∈ Q faire
    marge_min := −∞
    taille := 0
    k := NOMBRE_BOUCLES
    tant que (k ≥ 1) faire
      n := nombre_iterations[k]
      c := n * maj_echeance^X[k,q]
      c := c + (n−1) * maj_echeance_correction_intra^X[k,q]
      marge := echeance[k] − c
      c := c + maj_echeance_correction_inter^X[k,q]
      si (marge < marge_mini − c) faire
        taille := taille + 1
        t_s^X_pile[taille,q] := marge_mini
        t_s^X_pile_indice[taille,q] := k + 1
        marge_mini := marge
      sinon faire
        marge_mini := marge_mini − c
      fin si
      k := k − 1
    fin tant que
    t_s^X_pile_taille[q] := taille
    t_s^X[q] := marge_mini
  fin pour tout
fin Initialiser_t_s^X
```

FIG. 4.11: Algorithme d'initialisation des variables globales.

Calcul de t_s^{av}

Sur la base des résultats du corollaire 4.1, les tables sont calculées de la façon suivante dans le cas particulier de t_s^{av}.

$$\begin{aligned}
\texttt{maj_action}^{av}[i,q] &:= C^{av}(\alpha_0[i,i+1],q) - C^{av}(\alpha_0(i+1),q) = C^{av}(\alpha_0(i),q) \\
\texttt{maj_echeance}^{av}[k,q] &:= C^{av}(\alpha_k,q) \\
\texttt{maj_echeance_correction_intra}^{av}[k,q] &:= C^{av}(b_k a_k,q) - C^{av}(b_k,q) - C^{av}(a_k,q) = 0 \\
\texttt{maj_echeance_correction_inter}^{av}[k,q] &:= C^{av}(b_k a_{k+1},q) - C^{av}(b_k,q) - C^{av}(a_{k+1},q) = 0.
\end{aligned}$$

Puisque les valeurs présentent dans les tables `maj_echeance_corrections_intra` et `maj_echeance_corrections_inter` sont toutes nulles, nous pouvons simplifier l'algorithme *Initialiser_t_s^av*. La figure 4.14 donne la version simplifiée de *Initialiser_t_s^av*.

```
Calculer_t_s^X(i,k)
  pour tout q ∈ Q faire
    taille := t_s^X_pile_taille[q]
    si (k > t_s^X_pile_indice[taille,q]) faire
      MAJ_Boucle_t_s^X(k,q)
    sinon faire
      MAJ_Action_t_s^X(i,q)
    fin si
  fin pour tout
fin Calculer_t_s^X
```

FIG. 4.12: Algorithme principal de calcul de t_s^X.

```
MAJ_Boucle_t_s^X(k,q)
  taille := t_s^X_pile_taille[q]
  t_s^X[q] := t_s^X_pile[taille,q]
  t_s^X_pile_taille[q] := taille − 1
fin MAJ_Boucle_t_s^X

MAJ_Action_t_s^X(i,q)
  t_s^X[q] := t_s^X[q] + maj_action^X[i]
fin MAJ_Action_t_s^X
```

FIG. 4.13: Algorithmes de mise à jour de la fonction d'ordonnancement.

Calcul de t_s^{sf}

Dans le cas de la politique de gestion de qualité sûre, les tables pré-calculées sont définies de la façon suivante

$$\begin{aligned}
\texttt{maj_action}^{sf}[i,q] &:= C^{sf}(\alpha_0[i,i+1],q) - C^{sf}(\alpha_0(i+1),q) \\
&= C^{wc}(\alpha_0(i),q) + C^{wc}(\alpha_0(i+1),q_{min}) - C^{wc}(\alpha_0(i+1),q) \\
\texttt{maj_echeance}^{sf}[k,q] &:= C^{sf}(\alpha_k,q) \\
\texttt{maj_echeance_intra}^{sf}[k,q] &:= C^{sf}(b_k a_k,q) - C^{sf}(b_k,q) - C^{sf}(a_k,q) \\
&= C^{wc}(a_k,q_{min}) - C^{wc}(a_k,q) \\
\texttt{maj_echeance_inter}^{sf}[k,q] &:= C^{sf}(b_k a_{k+1},q) - C^{sf}(b_k,q) - C^{sf}(a_{k+1},q) \\
&= C^{wc}(a_{k+1},q_{min}) - C^{wc}(a_{k+1},q).
\end{aligned}$$

Calcul de t_s^{sp}

Nous avons vu comment calculer efficacement la valeur des fonction d'ordonnancement t_s^{av} et t_s^{sf} sur tout point de contrôle dans le cas d'un ordonnanceur statique. La fonction d'ordonnancement t_s^{sp} correspondant à la politique de gestion de qualité simple peut être obtenue facilement à partir des fonctions d'ordonnancement t_s^{av} et t_s^{sf}, puisque nous savons que la fonction d'ordonnancement t_s^{sp} vérifie $t_s^{sp} = \mathbf{min}\ t_s^{av}, t_s^{sf}$ (voir proposition 3.8 de la section 3.3.2).

```
Initialiser_t_s^av()
  pour tout q ∈ Q faire
    marge_min := −∞
    taille := 0
    k := NOMBRE_BOUCLES
    tant que (k ≥ 1) faire
      n := nombre_iterations[k]
      c := n * maj_echeance^av[k,q]
      marge := echeance[k] − c
      si (marge < marge_mini − c) faire
        taille := taille + 1
        t_s^av_pile[taille,q] := marge_mini
        t_s^av_pile_indice[taille,q] := k + 1
        marge_mini := marge
      sinon faire
        marge_mini := marge_mini − c
      fin si
      k := k − 1
    fin tant que
    t_s^av_pile_taille[q] := taille
    t_s^av[q] := marge_mini
  fin pour tout
fin Initialiser_t_s^av
```

FIG. 4.14: Initialisation dans le cas de la politique de gestion de qualité moyenne

EXEMPLE 4.2 *Considérons l'ensemble des niveaux de qualité $Q = \{0,1\}$, l'ensemble d'actions $A = \{a,b,c,d,e,f\}$ et un contrôleur basé sur un ordonnancement statique $\alpha = abcdef$. La fonction d'échéance $D : A \to \mathbb{R}^+ \cup \{+\infty\}$ et les durées d'exécution $C^{av} : A \times Q \to \mathbb{R}^+$ et $C^{wc} : A \times Q \to \mathbb{R}^+$ sont données par la figure 4.16. Nous avons* $\text{critique}_D(\alpha) = \{2,6\}$. *Les valeurs pré-calculées sont données dans la figure 4.17. La figure 4.18 donne l'algorithme correspondant à l'application contrôlée. Dans celui-ci, les fonctions a, b, c, d, e, g représentent l'implémentation des actions Action_a, Action_b, Action_c, Action_d, Action_e, Action_f, Action_g. Elles prennent en argument le niveau de qualité q auquel elle doivent s'exécuter. Nous supposons de plus qu'elles mettent à jour le temps-réel t.*

4.5 Calcul de t_s^{mx}

Dans la proposition suivante, nous montrons que la politique de gestion de qualité mixte n'est issue d'une fonction de durée d'exécution à mise-à-jour rapide. La preuve est faite en exhibant un contre-exemple dans lequel nous considérons un système paramétré incertain constitué de trois actions et deux niveaux de qualité.

$Calculer\_t_s^{sp}(i,k)$		
$\quad Calculer\_t_s^{av}(i,k)$		
$\quad Calculer\_t_s^{sf}(i,k)$		$Initialiser\_t_s^{sp}()$
$\quad$ **pour tout** $q \in Q$ **faire**		$\quad Initialiser\_t_s^{av}()$
$\quad\quad t_s^{sp}[q]$	$:=$	$\quad Initialiser\_t_s^{sf}()$
$\mathbf{min}(t_s^{av}[q], t_s^{sf}[q])$		**fin** $Initialiser\_t_s^{sp}$
$\quad$ **fin pour tout**		
fin $Calculer\_t_s^{sp}$		

FIG. 4.15: Algorithme de calcul de t_s^{sp} à partir de t_s^{av} et t_s^{sf}.

action x	$D(x)$	$C^{av}(x,0)$	$C^{av}(x,1)$	$C^{wc}(x,0)$	$C^{wc}(x,1)$
a	$+\infty$	1	2	3	4
b	10	2	3	4	7
c	$+\infty$	1	4	2	6
d	$+\infty$	4	6	10	14
e	60	3	6	10	12
f	50	10	20	20	30

FIG. 4.16: Echéances et durées d'exécution.

PROPOSITION 4.5 *La fonction de durée d'exécution mixte C^{mx} n'est pas à mise-à-jour rapide.*

Preuve de la proposition : Soit un système paramétré incertain $SPI(C) = (G, C^{wc}, D, Q; C)$, $G = (A, \prec)$, composé de trois actions $A = \{ a, b, c \}$ et deux niveaux de qualité $Q = \{ 0, 1 \}$, tel que le graphe de précédence soit vide, c'est-à-dire $\prec = \emptyset$. Nous supposons que les durées d'exécution pire-cas sont données par la figure 4.19.

Si la politique de C^{mx} était une fonction de durée d'exécution à mise-à-jour rapide, nous aurions en particulier :

$$C^{mx}(ab,1) - C^{mx}(b,1) = C^{mx}(abc,1) - C^{mx}(bc,1).$$

Puisque $C^{mx} = C^{av} + \delta^{max}$, nous aurions :

$$\begin{aligned} C^{av}(ab,1) + \delta^{max}(ab,1) - C^{av}(b,1) - \delta^{max}(b,1) &= C^{av}(abc,1) + \delta^{max}(abc,1) - C^{av}(bc,1) - \delta^{max}(bc,1) \\ C^{av}(a,1) + \delta^{max}(ab,1) - \delta^{max}(b,1) &= C^{av}(a,1) + \delta^{max}(abc,1) - \delta^{max}(bc,1) \\ \delta^{max}(ab,1) - \delta^{max}(b,1) &= \delta^{max}(abc,1) - \delta^{max}(bc,1). \end{aligned}$$

Or nous avons :

$$\delta^{max}(ab,1) - \delta^{max}(b,1) = 3 - 2 = 1$$

et :

$$\delta^{max}(abc,1) - \delta^{max}(bc,1) = 9 - 9 = 0.$$

Ainsi, nous avons démontré que C^{mx} n'est pas une fonction de durée d'exécution à mise-à-jour rapide. □

NOMBRE_ACTIONS := 6
NOMBRE_BOUCLES := 2

echeance[k]	2	6
k	1	2

maj_action$^{av}[i,q]$	$q=0$	$q=1$
$i=1$	1	2
$i=2$	2	3
$i=3$	1	4
$i=4$	4	6
$i=5$	3	6
$i=6$	10	20

maj_action$^{sf}[i,q]$	$q=0$	$q=1$
$i=1$	3	1
$i=2$	4	3
$i=3$	2	2
$i=4$	10	12
$i=5$	10	-8
$i=6$	20	30

maj_echeance$^{av}[k,q]$	$q=0$	$q=1$
$k=1$	3	5
$k=2$	18	36

maj_echeance$^{sf}[k,q]$	$q=0$	$q=1$
$k=1$	7	4
$k=2$	42	46

maj_echeance2$^{sf}[k,q]$	$q=0$	$q=1$
$k=1$	0	8
$k=2$	0	0

FIG. 4.17: Valeur pré-calculées.

Puisque la fonction de durée d'exécution mixte C^{mx} n'est pas à mise-à-jour rapide, il n'est pas possible de déduire de façon simple la valeur de la fonction d'ordonnancement t_s^{mx} sur un point de contrôle donné à partir du point de contrôle précédent, comme nous l'avons fait pour les politique de gestion de qualité moyenne, sûre et simple. A chaque instant nous allons devoir mémoriser les marges par rapport à toutes les échéances potentiellement critiques, c'est-à-dire les valeurs $t_s^{mx}(\alpha,\theta)(k)$ pour tout $k \in \mathsf{critique}_D(\alpha)$. D'après les propositions 3.1 et 3.9 (voir chapitre 3), ces valeurs seront suffisantes pour calculer $t_s^{mx}(\alpha,\theta)$.

Nous savons que $t_s^{mx}(\alpha,\theta)(k) = t_s^{av}(\alpha,\theta)(k) - \delta^{max}({}^k\alpha,\theta)$. Puisque C^{av} est une fonction de durée d'exécution à mise-à-jour rapide, nous pouvons calculer efficacement $t_s^{av}(\alpha,\theta)(k)$. Ainsi, ce qui suit sera essentiellement consacré au calcul efficace la fonction δ^{max}, dans le cas d'une politique d'ordonnancement statique.

4.5.1 Calcul efficace de δ^{max} dans le cas d'une échéance unique

Dans un premier temps, nous allons voir comment calculer la fonction d'ordonnancement t_s^{mx} lorsque la fonction d'échéance $D : A_0 \to \mathbb{R}^+ \cup \{ +\infty \}$ est une fonction constante. Nous avons vu que dans la section 2.4 que, dans ce cas, la fonction d'ordonnancement s'écrit :

$$t_s^{mx}(\alpha,\theta) = t_s^{av}(\alpha,\theta) - \delta^{max}(\alpha,\theta).$$

Puisque C^{av} est une fonction de durée d'exécution à mise-à-jour rapide, il est facile de calculer la valeur $t_s^{av}(\alpha,\theta)$. Nous allons voir comment calculer efficacement δ^{max}.

$t := 0$
$Initialiser\_t_s^{sp}$
$Action\_a(Choisir\_q\_sp(0,0,t))$
$Action\_b(Choisir\_q\_sp(1,0,t))$
$Action\_c(Choisir\_q\_sp(2,1,t))$
$Action\_d(Choisir\_q\_sp(3,1,t))$
$Action\_e(Choisir\_q\_sp(4,1,t))$
$Action\_f(Choisir\_q\_sp(5,1,t))$

FIG. 4.18: Algorithme correspondant à l'application contrôlée.

action x	$C^{av}(x,q)$	$C^{wc}(x,q)$
a	q + 1	q + 3
b	q + 1	q + 3
c	q + 1	10q + 1

FIG. 4.19: Durées d'exécution moyennes et pire-cas.

Propriétés de la fonction δ^{max} relatives aux boucles

PROPOSITION 4.6 (calcul de δ^{max} par bloc) *Nous avons les résultats suivant sur la fonction δ^{max} présentée dans le chapitre 3 :*

1. $\delta^{max}(\alpha_1\alpha_2\ldots\alpha_m,\theta) = \mathbf{max}_{1\leq k\leq m}\ \delta^{max}(\alpha_k,\theta) + \sum_{l>k}\beta(\alpha_j,\theta)$.
2. *Pour tout $n>0$ nous avons* $\delta^{max}(\{\alpha\}^n,\theta) = \begin{cases} \delta^{max}(\alpha,\theta) \text{ si } \beta(\alpha,\theta)\leq 0 \\ \delta^{max}(\alpha,\theta) + (n-1)\beta(\alpha) \text{ sinon.}\end{cases}$

Preuve de la proposition :

1. Nous décomposons δ^{max} à l'aide des fonctions η et β définies dans la section 3.4.2 :

$$\begin{aligned}
\delta^{max}(\alpha_1\alpha_2\ldots\alpha_m,\theta) &= \mathbf{max}_{1\leq i\leq|\alpha_1\alpha_2\ldots\alpha_m|}\ \eta((\alpha_1\alpha_2\ldots\alpha_m)(i),\theta) + \beta((\alpha_1\alpha_2\ldots\alpha_m)^{i+1},\theta) \\
&= \mathbf{max}_{1\leq k\leq m}\ \mathbf{max}_{1\leq i\leq|\alpha_k|}\ \eta(\alpha_k(i),\theta) + \beta({\alpha_k}^{i+1},\theta) + \sum_{l>k}\beta(\alpha_l,\theta) \\
&= \mathbf{max}_{1\leq k\leq m}\ (\mathbf{max}_{1\leq i\leq|\alpha_k|}\ \delta({\alpha_k}^{i},\theta)) + \sum_{l>k}\beta(\alpha_l,\theta) \\
&= \mathbf{max}_{1\leq k\leq m}\ \delta^{max}(\alpha_k,\theta) + \sum_{l>k}\beta(\alpha_l,\theta).
\end{aligned}$$

2. La démonstration du second point est un cas particulier du premier résultat. En effet, d'après ce qui précède, pour tout $n>0$, nous avons :

$$\delta^{max}(\{\alpha\}^n,\theta) = \mathbf{max}_{1\leq k\leq n}\ \delta^{max}(\alpha,\theta) + \sum_{l>k}\beta(\alpha,\theta).$$

Si $\beta(\alpha,\theta) \leq 0$, alors le maximum de l'expression précédente est atteint pour $k = n$, c'est-à-dire $\delta^{max}(\{\alpha\}^n,\theta) = \delta^{max}(\alpha,\theta)$. Dans le cas où nous avons $\beta(\alpha,\theta) > 0$, la maximum est atteint pour $k = 1$, c'est-à-dire $\delta^{max}(\{\alpha\}^n,\theta) = \delta^{max}(\alpha,\theta) + (n-1)\beta(\alpha,\theta)$. □

La proposition suivante permet de calculer la valeur de la fonction δ^{max} sur un point de contrôle d'une boucle $\{\alpha\}^n$. Nous l'exprimons en fonction de la valeur de la fonction δ^{max} sur des points de contrôle du corps de boucle α, et de la valeur $\beta(\alpha,\theta)$.

PROPOSITION 4.7 (calcul δ^{max} pour une boucle) *Soit $\{\alpha\}^n$ une boucle. Dans la suite, p et r désignent respectivement le résultat et le reste de la division entière de $k-1$ par $|\alpha|$, c'est-à-dire $k-1 = p|\alpha| + r$ et $0 \leq r < |\alpha|$.*

1. *Si $\beta(\alpha,\theta) \leq 0$, alors la valeur de la fonction δ^{max} est donnée, pour tout suffixe $(\{\alpha\}^n)^k$ de $\{\alpha\}^n$, par :*
$$\delta^{max}((\{\alpha\}^n)^k,\theta) = \begin{cases} \delta^{max}(\alpha,\theta) \text{ si } p < n-1 \\ \delta^{max}(\alpha^{r+1},\theta) \text{ si } p = n-1. \end{cases}$$

2. *Si $\beta(\alpha,\theta) > 0$, alors la valeur de la fonction δ^{max} est donnée, pour tout suffixe $(\{\alpha\}^n)^k$ de $\{\alpha\}^n$, par :*
$$\delta^{max}((\{\alpha\}^n)^k,\theta) = \begin{cases} \delta^{max}(\alpha) + (n-p-1)\beta(\alpha,\theta) \text{ si } r = 0 \\ \delta^{max}(\alpha^{r+1},\theta) \text{ si } r > 0 \text{ et } p = n-1 \\ \textbf{max}\{\ \delta^{max}(\alpha^{r+1},\theta) + (n-p-1)\beta(\alpha,\theta),\ \delta^{max}(\alpha) + (n-p-2)\beta(\alpha,\theta)\ \} \text{ sinon.} \end{cases}$$

Preuve de la proposition : La preuve de la proposition est fait principalement appel aux résultats énoncés dans la proposition 4.6. Soit $(\{\alpha\}^n)^k$ de $\{\alpha\}^n$ un suffixe de la boucle $\{\alpha\}^n$. Nous écrivons ce suffixe en faisant apparaître les entiers p et r :

$$\alpha^{r+1}\{\alpha\}^{n-p-1}.$$

Ainsi, grâce au premier résultat de la proposition 4.6 nous avons :

$$\begin{aligned} \delta^{max}((\{\alpha\}^n)^k,\theta) &= \delta^{max}(\alpha^{r+1}\{\alpha\}^{n-p-1},\theta) \\ &= \textbf{max}\{\ \delta^{max}(\alpha^{r+1},\theta) + (n-p-1)\beta(\alpha,\theta),\ \delta^{max}(\{\alpha\}^{n-p-1},\theta)\ \}. \end{aligned}$$

1. Supposons tout d'abord que $\beta(\alpha,\theta) \leq 0$. Lorsque $n-p-1 > 0$, c'est-à-dire $p < n-1$, le second résultat de la proposition 4.6 permet d'écrire :
$$\begin{aligned} \delta^{max}((\{\alpha\}^n)^k,\theta) &= \textbf{max}\{\ \delta^{max}(\alpha^{r+1},\theta) + (n-p-1)\beta(\alpha,\theta),\ \delta^{max}(\{\alpha\}^{n-p-1},\theta)\ \} \\ &= \textbf{max}\{\ \delta^{max}(\alpha^{r+1},\theta) + (n-p-1)\beta(\alpha,\theta),\ \delta^{max}(\alpha,\theta)\ \}. \end{aligned}$$
Puisque $\delta^{max}(\alpha^{r+1},\theta) \leq \delta^{max}(\alpha,\theta)$ et $\beta(\alpha,\theta) \leq 0$, nous pouvons conclure que, lorsque $\beta(\alpha,\theta) \leq 0$ et $p < n-1$:
$$\delta^{max}((\{\alpha\}^n)^k,\theta) = \delta^{max}(\alpha,\theta).$$
Lorsque $p = n-1$, nous avons $\{\alpha\}^{n-p-1} = \varepsilon$, et donc :
$$\delta^{max}((\{\alpha\}^n)^k,\theta) = \delta^{max}(\alpha^{r+1},\theta).$$

2. Supposons que $\beta(\alpha,\theta) \leq 0$. Si $r = 0$, nous avons :

$$\begin{aligned}\delta^{max}((\{\alpha\}^n)^k,\theta) &= \delta^{max}(\{\alpha\}^{n-p},\theta)\\ &= \delta^{max}(\alpha,\theta)+(n-p-1)\beta(\alpha,\theta).\end{aligned}$$

Si $r > 0$ et $p = n-1$, nous avons :

$$\delta^{max}((\{\alpha\}^n)^k,\theta) = \delta^{max}(\alpha^{r+1},\theta).$$

Si $p < n-1$, nous avons :

$$\begin{aligned}\delta^{max}((\{\alpha\}^n)^k,\theta) &= \mathbf{max}\{\ \delta^{max}(\alpha^{r+1},\theta)+(n-p-1)\beta(\alpha,\theta),\ \delta^{max}(\{\alpha\}^{n-p-1},\theta)\ \}\\ &= \mathbf{max}\{\ \delta^{max}(\alpha^{r+1},\theta)+(n-p-1)\beta(\alpha,\theta),\ \delta^{max}(\alpha,\theta)+(n-p-2)\beta(\alpha,\theta)\ \}.\end{aligned}$$

Valeurs pré-calculées

Soit $SPI(C) = (GH, C^{wc}, D, Q; C)$ un système paramétré incertain dont le graphe de précédence est hiérarchique, $GH = (GS, \prec, \nu)$. Soit *Best_Sched*^{mx} une politique d'ordonnancement statique basée sur l'ordonnancement $\alpha = \{\alpha_1\}^{n_1}\{\alpha_2\}^{n_2}\ldots\{\alpha_m\}^{n_m}$ du graphe hiérarchique GH. Nous notons α_0 l'ordonnancement des corps de boucle, c'est-à-dire $\alpha_0 = \alpha_1\alpha_2\ldots\alpha_m$. Avant de présenter les algorithmes qui permettent de calculer efficacement la valeur de δ^{max} en tout point de contrôle de l'ordonnancement α, nous définissons un ensemble de valeurs statiquement pré-calculées. Elle nous serviront à d'accélérer de manière significative le calcul de δ^{max}. Le principe est de calculer δ^{max} à deux niveaux de granularité : le niveau de granularité des boucles, et le niveau de granularité des actions, qui est un raffinement du niveau précédent.

`NOMBRE_BOUCLES`	nombres de boucles composant l'ordonnancement α
`delta_max_action`$[i,q]$	valeur δ^{max} sur un point de contrôle d'un corps de boucle
`delta_max`$[k,q]$	valeur δ^{max} du $k^{\text{ième}}$ corps de boucle
`beta`$[k,q]$	valeur β du $k^{\text{ième}}$ corps de boucle

FIG. 4.20: Valeurs pré-calculées statiquement permettant le calcul efficace de δ^{max}.

Tout d'abord `NOMBRE_BOUCLES` représente le nombre de boucles qui constitue l'ordonnancement α, c'est-à-dire :

$$\texttt{NOMBRE\_BOUCLES} := m.$$

Les tables `delta_max_action` et `delta_max` contiennent toutes les deux un ensemble de valeurs de la fonction δ^{max} pré-calculées. Ainsi, la table `delta_max` contient la valeur de δ^{max} pour le $k^{\text{ième}}$ corps de boucle, c'est-à-dire :

$$\texttt{delta\_max}[k,q] := \delta^{max}(\alpha_k, q).$$

La table `delta_max_action` contient des valeurs de δ^{max} correspondant à des points de contrôle des corps de boucles. Plus précisément, si i est un indice dans l'ordonnancement α_0, c'est-à-dire

$1 \leq i \leq |\alpha_0|$. Soit k le plus grand indice tel que $i > |\alpha_1 \dots \alpha_k|$, alors `delta_max_action`$[i,q]$ est défini par :

$$\texttt{delta\_max\_action}[i,q] := \delta^{max}(\alpha_{k+1}{}^{i-|\alpha_1 \dots \alpha_k|}, q).$$

L'algorithme *Calculer*_`delta_max_action` donné dans la figure 4.21 permet de calculer efficacement les valeurs de la table `delta_max_action`. Il n'est pas utilisé lors de l'exécution de l'application contrôlée, mais il est seulement utilisé lors de la génération du contrôleur afin de pré-calculer la table `delta_max_action`. Il utilise l'ensemble *DERNIERES_ACTIONS* des indices correspondant aux dernières actions des corps de boucle dans l'ordonnancement α_0, c'est-à-dire $DERNIERES\_ACTIONS = \{|\alpha_1 \dots \alpha_k| \mid 1 \leq k \leq m\}$. Pour finir, la table `beta` contient simplement les valeurs de la fonction β pour tous les corps de boucle, c'est-à-dire :

$$\texttt{beta}[k,q] := \beta(\alpha_k, q) = \sum_{1 \leq i \leq |\alpha_k|} \beta(\alpha_k(i), q).$$

*Calculer*_`delta_max_action`(α_0)
  $DERNIERES\_ACTIONS := \{|\alpha_1 \dots \alpha_k| \mid 1 \leq k \leq m\}$
  pour tout $q \in Q$ **faire**
  $i := |\alpha_0|$
  $delta\_max := -1$
  $somme\_beta := 0$
  tant que $(i \geq 1)$ **faire**
    si $(i \in DERNIERES\_ACTIONS)$ **faire**
      $delta\_max := -1$
      $somme\_beta := 0$
    fin si
    $a = \alpha(k)$
    $delta := C^{wc}(a,q) - C^{av}(a,q) + somme\_beta$
    si $(delta > delta\_max)$ **faire**
      $delta\_max := delta$
    fin si
    `delta_max_action`$[i,q] := delta\_max$
    $somme\_beta = somme\_beta + C^{wc}(a, q_{min}) - C^{av}(a,q)$
    $k := k-1$
  fin tant que
fin *Calculer*_`delta_max_action`

FIG. 4.21: Algorithme de calcul de la table `delta_max_action`.

Algorithmes de calcul de δ^{max}

Outres les valeurs pré-calculées, le calcul de la fonction δ^{max} requière les tableaux δ^{max}, δ^{max}_pile,$\delta^{max}\_pile\_indice$, $\delta^{max}\_pile\_taille$ et $some\_beta\_pile$, afin de conserver certaines valeurs

calculées dynamiquement. Le contenu de ces tableaux est initialisé par la fonction *Initialiser*_δ^{max} (figure 4.23).

Le tableau δ^{max} de dimension $|Q|$ contient, à tout point de contrôle, la valeur de la fonction δ^{max} de la séquence d'actions qui reste à exécuter. Les tableaux δ^{max}*_pile*, δ^{max}*_pile_indice*, et *some_beta_pile* sont utilisés pour empiler certaines valeurs calculées au démarrage de l'application. Leur dimension maximale est $m \times |Q|$, et l'indice de la prochaine valeur à dépiler est donnée par le tableau de δ^{max}*_pile_taille* de dimension $|Q|$.

Plus précisément, la pile δ^{max}*_pile* contient les différentes valeurs de la fonction δ^{max} au niveau de granularité des boucles. Le tableau δ^{max}*_pile_indice* contient l'indice des boucle après lesquelles la fonction δ^{max} change de valeur. Enfin, la pile *some_beta_pile* contient la valeur de la fonction β pour les boucles

$\delta^{max}[q]$	valeur de δ^{max} sur le point de contrôle courant
δ^{max}*_pile*$[i,q]$	pile de valeurs de δ^{max} au niveau de granularité des boucles
δ^{max}*_pile_indice*$[i,q]$	indice des boucles précédent les changements de valeur
δ^{max}*_pile_taille*$[q]$	taille des piles
some_beta_pile$[i,q]$	somme des valeurs β des boucles qui suivent la boucle qui maximise δ

FIG. 4.22: Tableaux utilisés pour le calcul de δ^{max}.

Ces valeurs étant calculées, le calcul de la fonction δ^{max} pour n'importe quel point de contrôle est réalisé par la fonction *Calculer*_δ^{max}, dont l'argument est un triplet de la forme (i,k,n), qui décrit le point de contrôle courant dans l'ordonnancement α. Plus précisément, i représente la position de la prochaine action à exécuter dans l'ordonnancement α_0, k l'indice de la boucle en cours d'exécution, et n l'indice de l'itération en cours, la première itération ayant pour indice 1. Autrement dit, le point de contrôle défini par (i,k,n) signifie que la séquence d'actions $\{\alpha_1\}^{n_1} \ldots \{\alpha_{k-1}\}^{n_{k-1}} \{\alpha_k\}^{n-1} {\alpha_k}^{i-|\alpha_1 \ldots \alpha_{k-1}|}$ correspond aux actions qui ont déjà été exécutées.

La fonction *Calculer*_δ^{max} fait appel à la fonction *MAJ_Action*_δ^{max} (figure 4.25) qui permet de raffiner δ^{max} lorsque la valeur maximal de δ est atteinte sur le point de contrôle courant de la boucle courante.

```
Initialiser_δ^max()
  pour tout q ∈ Q faire
    somme_beta := 0
    delta_max := −1
    taille := 0
    k := NOMBRE_BOUCLES
    tant que (k ≥ 1) faire
      n := nombre_iterations[k]
      β := delta_max[k,q]
      si (β > 0) faire
        delta := delta_max[k,q] + (n−1) ∗ β + somme_beta
      sinon faire
        delta := delta_max[k,q] + somme_beta
      fin si
      si (delta > delta_max) faire
        delta_max := delta
        taille := taille + 1
        δ^max_pile[taille,q] := delta_max
        δ^max_pile_indice[taille,q] := k
        somme_beta_pile[taille,q] := somme_beta
      fin si
      somme_beta := somme_beta + n ∗ β
      k := k − 1
    fin tant que
    δ^max[q] := delta_max
    δ^max_taille[q] := taille
  fin pour tout
fin Initialiser_δ^max
```

FIG. 4.23: Initialisation des valeurs pour le calcul de δ^{max}.

```
Calculer_δ^max(i,k,n)
  pour tout q ∈ Q faire
    taille = δ^max_pile_taille[q]
    si (k = δ^max_pile_indice[taille,q]) faire
      MAJ_δ^max(i,k,n,q)
      si (taille ≥ 2) faire
        taille := taille − 1
        si (δ^max[q] < δ^max_pile[taille,q]) faire
          δ^max[q] := δ^max_pile[taille,q]
          δ^max_pile_taille[q] := taille
        fin si
      fin si
    fin si
  fin pour tout
fin Calculer_δ^max
```

FIG. 4.24: Calcul de δ.

```
MAJ_Action_δ^max(i,k,n,q)
  δ^max := delta_max_action[i,q]
  β := beta[k,q]
  somme_beta := somme_beta_pile[δ^max_pile_taille[q],q]
  si (β > 0) faire
    δ1 := δ^max + (nombre_iterations[k] − n) ∗ β + somme_beta
    si (n < nombre_iterations[k]) faire
      δ2 := delta_max[k,q] + (nombre_iterations[k] − n − 1) ∗ β +
somme_beta
    sinon faire
      δ2 := 0
    fin si
    δ^max[q] := max{ δ1, δ2 }
  sinon faire
    si (n = nombre_iterations[k]) faire
      δ^max[q] := δ^max + somme_beta
    fin si
  fin si
fin MAJ_Action_δ^max
```

FIG. 4.25: Calcul de δ^{max} d'un point de contrôle d'une boucle

CHAPITRE 5

Résultats expérimentaux

Le travail de thèse présenté dans ce manuscrit a été mené à partir d'une application cible. Il s'agit d'une application d'encodage vidéo. Dans un premier temps, nous ferons une présentation complète de l'application vidéo considérée. Nous expliquerons ainsi les principes fondateurs des algorithmes d'encodage vidéo les plus utilisés. Nous détaillerons également le modèle de l'application que nous avons choisi. La deuxième partie de ce chapitre concernera les résultats expérimentaux que nous avons obtenus. Ces résultats visent à montrer le bien fondé de notre approche.

5.1 Application cible : encodeur vidéo MPEG4

5.1.1 Généralités sur la compression vidéo de type MPEG4

La compression de données consiste à encoder des données d'une manière particulière qui permet de réduire l'espace nécessaire à son stockage, ou le temps nécessaire à sa transmission. Ainsi, le terme *encodage* est fréquemment utilisé pour désigner le processus de compression. En informatique, les données d'entrée seront données par une suite de bits $\mathbf{b} = b_1 \dots b_n$, $b_i \in \{ 0,1 \}$. Si l'on désigne par C le processus de compression, le but est que la suite de bits $\mathbf{b}' = b'_1 \dots b'_m = C(b_1 \dots b_n)$ soit telle que $m \leq n$. Le taux de compression sera donné par le rapport $\frac{n-m}{n}$. On distingue deux grandes familles de compressions de donnés.

Les compressions conservatives. Avec une compression *conservative*, il est possible de retrouver les données d'entrée à partir des données compressées, c'est-à-dire la fonction C est injective.

Ce type de compression est en général utilisé compresser des données pour lesquels on accepte aucune modification (texte, exécutable binaire, etc...).

Les compression non-conservatives. Avec compression *non-conservative*, il n'est pas possible de retrouver dans tous les cas les donnée d'entrée à partir des données compressées. Dans ce cas, la fonction C n'est pas injective et nous reconstruisons seulement une approximation des données d'entrée à partir des données compressées. Le but est évidemment que cette approximation soit la meilleure possible, et cela dépend fortement du type de données que l'on doit compresser. Plus exactement, à partir de la fonction de décodage D et d'un taux de compression τ, nous cherchons à construire une fonction C_τ telle que $D(C_\tau(\mathbf{b}))$ soit le plus "proche" possible de $\mathbf{b}$, et tel que le taux de compression soit de τ.

La plupart des algorithme de compression, qu'ils soient conservatifs ou non-conservatifs, exploitent les redondances, c'est-à-dire les corrélations, entre les données d'entrée. La façon dont ces redondances et corrélations apparaissent dépend fortement des données d'entrée et la façon dont elles sont encodées au départ. C'est pourquoi les algorithmes de compression les plus efficaces sont en général dédiés à un type d'entrée particulière.

Nous nous intéressons ici à un algorithme de compression vidéo, ou encodeur vidéo, de type MPEG4. Il s'agit d'une compression non-conservative dédiée à la vidéo. L'algorithme prend en entrée un flux vidéo, ou flux de *frames*, issues par exemple d'une caméra vidéo. Ce flux vidéo peut être décrit par un signal à trois dimensions, dont une des dimensions est le temps. Les *frames* fournissent donc un échantillonnage dans le temps du signal.

Le but est alors de générer un flux de bits, ou *bitstream*, qui respecte la norme d'encodage MPEG4. Le principe dans un encodage MPEG4 est de tirer partie de deux types de corrélations dans le flux de *frames* en entrée.

Les corrélations spatiales. Elles désignent toutes les corrélations que l'on trouve à l'intérieur même d'une *frame*. En général, les algorithmes de compression font apparaître ces corrélations grâce une représentation fréquentielle telle que la transformée de Fourier discrète. Cette représentation fréquentielle permet également, pour un taux de compression τ donné, de moduler les dégradations de la *frame* encodée en fonction des fréquences et leurs intensités, ce qui correspond bien à la façon dont l'œil humain perçoit les images.

Les corrélations temporelles. Elles désignent toutes les corrélations qui existent entre les *frames*. En général, on désigne par *estimation de mouvement* le processus qui permet trouver ces corrélations. Les standards de compression récents comme H264 [3],[1] ou encore MPEG4 [4],[2] permettent d'encoder différentes sortes de corrélations temporelles : entre deux *frames* consécutives dans le flux d'entrée, entre deux *frames* distantes de plusieurs *frames* dans le flux d'entrée, ou encore entre une et plusieurs autres *frames*. De tels standards permettent d'augmenter les taux de compression, pour une qualité d'image donnée, au prix d'algorithmes d'estimation de mouvement d'une compleéchantillonnagexité très importante.

Soit f la *frame* courante que l'on cherche à encoder. Il existe ainsi deux manières principales d'encoder f ou toute partie de f. Avec l'encodage de type INTRA , on ne fait pas référence à une autre *frame* que f. Ce type de codage n'exploite donc que les corrélations spatiales. Un encodage de

type INTRA sera utilisé, par exemple, pour encoder les *frames* qui correspondent aux changements de plan dans de la vidéo.

L'encodage de type INTER permet de faire référence à une ou plusieurs *frames* autres que f. Les *frames* auquel l'encodage INTER fait référence peuvent être aussi bien située temporellement avant ou après f. L'encodage de type INTER exploite à la fois les corrélations spatiales du flux d'entrée et les corrélations temporelles.

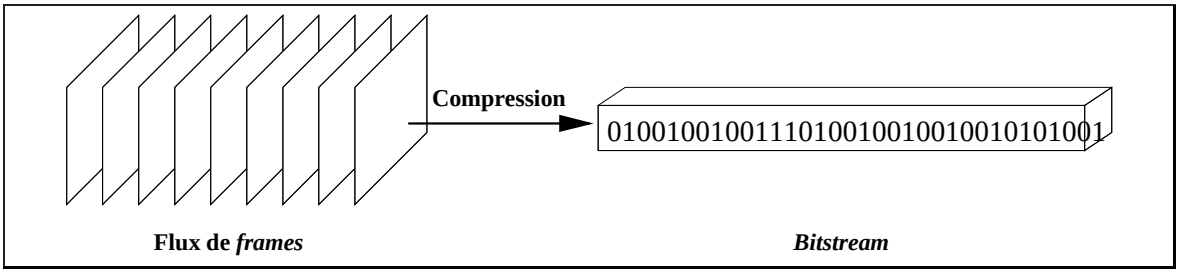


FIG. 5.1: Entrées et sorties du processus de compression.

5.1.2 Présentation de l'application

Le travail développé ici est motivé par une application réelle. Il s'agit de l'encodeur vidéo présent dans un visiophone développé chez STMicroelectronics. Cet encodeur vidéo est un *system-on-chip* capable de réaliser l'encodage en temps-réel d'un flux vidéo, dans un contexte embarqué tel que la téléphonie mobile. Les traitements vidéo et en particulier d'encodage vidéo demandent beaucoup de puissance de calcul. Ainsi, le système est basé sur une architecture parallèle et des accélérations matérielles. Le logiciel est donc écrit spécifiquement pour cette plate-forme d'exécution.

Il n'a pas été possible d'avoir accès à l'intégralité de l'application, ainsi qu'à une technologie de simulation d'une telle application. Afin de mener à bien nos expérimentations, nous avons fait le choix d'utiliser un logiciel plus générique, ainsi qu'une plate-forme — un basée sur un processeur XiRisc [7] — pour laquelle nous avons une technologie de simulation rapide. Dans les simulations, la cadence du processeur a été artificiellement gonflée à 11.250 GHz pour plusieurs raisons. D'abord, les plates-formes dédiées à l'encodage vidéo ont souvent une puissance de calcul très importante, du fait de l'utilisation d'accélérations matérielles et d'unités de calcul parallèles. Ensuite, le code source ainsi que le code machine produits par le compilateur ne sont pas très optimisés. Pour finir, nous avons fait le choix de faire nos essais sur une taille de frame et un débit de frame assez important, puisque la puissance des plates-formes embarquées est en constante augmentation.

5.1.3 Modélisation de l'application

Nous donnons ici le modèle de l'encodeur vidéo que nous avons utilisé pour les résultats expérimentaux présenté dans ce chapitre. Le modèle est un système paramétré incertain hiéarchique $SPIE(C,n) = (GH, C, D, Q, C^{wc}, C^{av}, n)$.

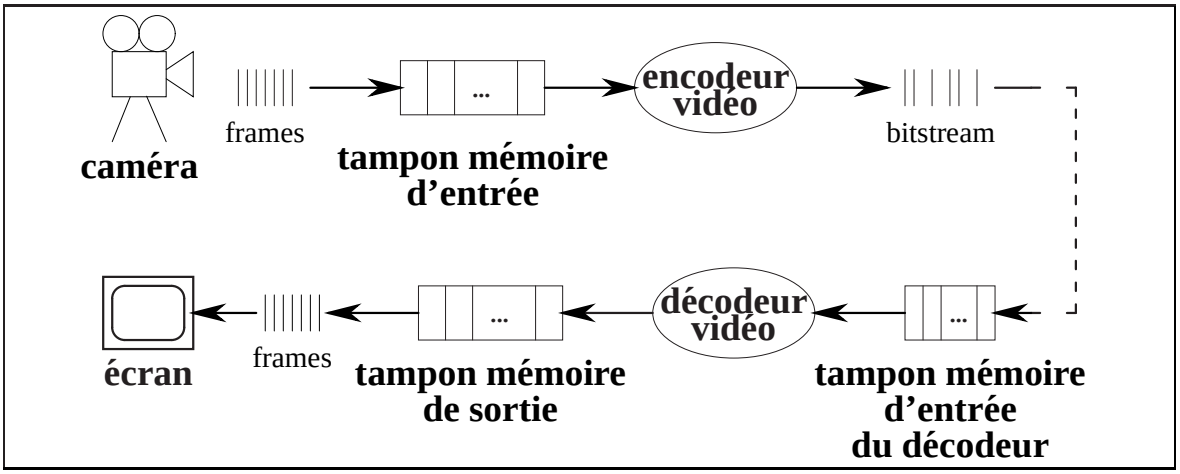


FIG. 5.2: .

Graphe hiérarchique

Le graphe hiérarchique *GH* est représenté dans la figure 5.5. Soit f la *frame* en entrée de l'algorithme. La première étape de l'algorithme, *GrabPicture*, est un pré-traitement sur f qui permet, en particulier, de déterminer si f est une *frame* INTER ou INTRA. D'autres traitement sont effectués, comme par exemple la transformation d'un codage des couleurs de type RVB, issu de la caméra vidéo, en un codage de type YUV.

La suite de l'algorithme est constituée par des traitements qui ne se situe pas au niveau de f tout entier, mais au niveau *macro-bloc*. Un macro-bloc est une portion de f, un carré 16×16 pixels dans l'algorithme considéré ici. Tout d'abord, le macro-bloc est récupéré dans la *frame* f à l'aide de *GrabMacroBlock*. Un macro-bloc 16×16 pixels nécessite 384 coefficients entiers pour être encodé.

Le traitement suivant est appelé estimation compensation de mouvement (*MotionEstimationCompensation*). Si la *frame* courante est de type INTRA, l'estimation de mouvement ne fait rien. Si la *frame* courante est de type INTER, nous déterminons si le macro-bloc courant est de type INTRA ou de type INTER. Dans ce dernier cas, l'estimation de mouvement calcule aussi son vecteur de mouvement, c'est-à-dire le vecteur entre la position du macro-bloc courant et la position du macro-bloc le plus ressemblant dans la *frame* précédente. Cette position peut être calculée au pixel près, ou au demi-pixel en utilisant des algorithmes d'interpolation d'image, ce qui améliore la qualité d'image pour un taux de compression donné. Le critère de ressemblance le plus utilisé est la somme des valeur absolues des différences (ou *SAD*), calculée sur la luminance uniquement. Plutôt que d'utiliser une recherche exhaustive du macro-bloc le plus ressemblant, qui est évidemment très coûteuse même pour une fenêtre de recherche de 16×16 pixels, des heuristiques sont utilisées. Une fois le vecteur de mouvement calculé, la compensation de mouvement consiste à calculer la différence entre le macro-bloc courant et le macro-bloc le plus ressemblant. Dans le cas d'un macro-bloc de type INTER, ce sont les coefficients issues de cette différence qui sont transmis aux étapes de calcul suivantes.

Le traitement suivant est la transformée de Fourier discrète à deux dimensions (*DCT*). Pour des

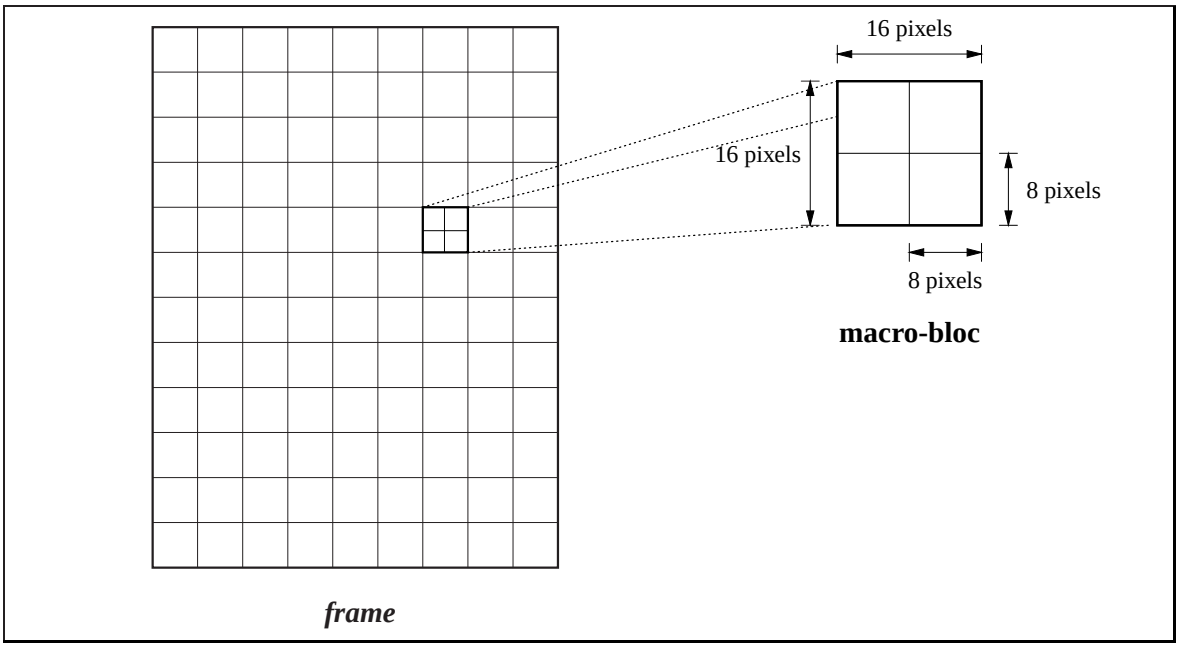


FIG. 5.3: *Frame*, macro-blocs et blocs.

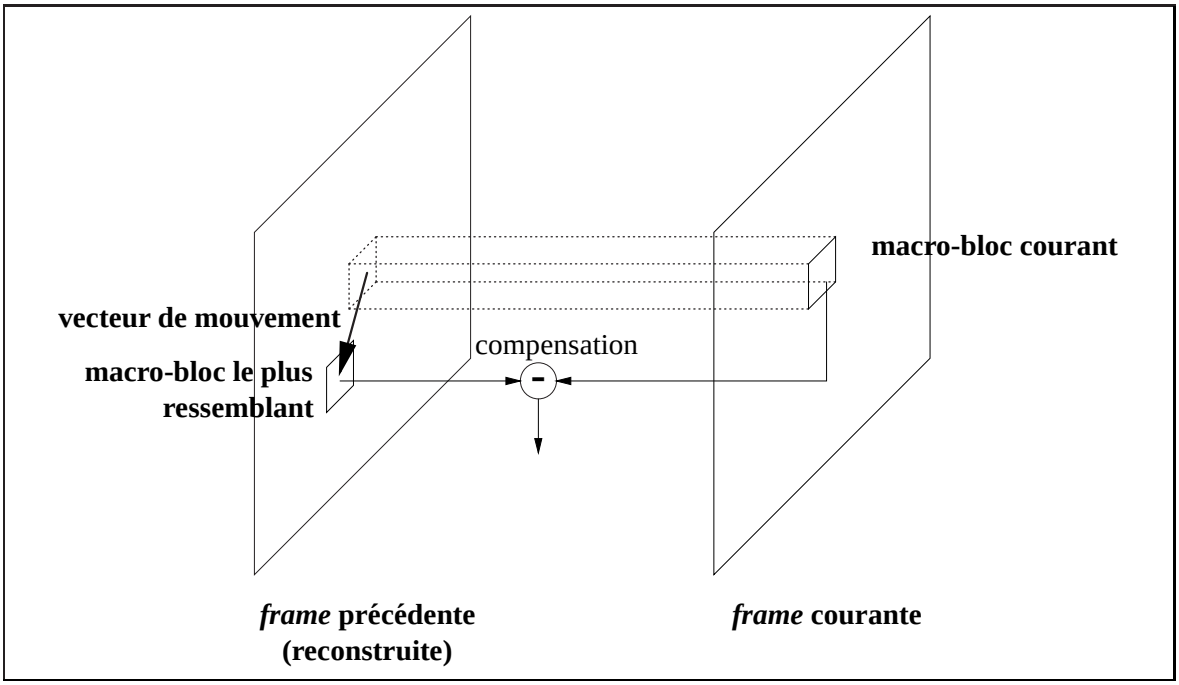


FIG. 5.4: Estimation de mouvement et compensation de mouvement.

raisons techniques, elle est implémentée pour des blocs d'image de 8×8 pixels. Le macro-bloc courant est donc découpé en quatre blocs de 8×8 (voir figure 5.4), sur lesquels sont appliqués la transformée de Fourier discrète. A partir des coefficients correspondant au macro-bloc courant, nous obtenons autant de coefficients avec la transformée de Fourier. Il s'agit donc d'une autre représentation des coefficients de départ, qui correspond à un changement de base. Les coefficients obtenus ne correspondent plus au différents pixels qui compose le macro-bloc, mais aux différentes fréquences qui compose le macro-bloc.

Le traitement suivant, appelé *quantization* (*Quant*), est une opération qui consiste à faire diviser les coefficients issus de la transformée de Fourier, de façon à réduire leur variabilité. Les coefficients sont plus ou moins affectés en fonction de la fréquence à laquelle ils correspondent, et en fonction du taux de compression qui est visé. Puisque ces divisions sont effectuées dans les entiers, cette étape de l'algorithme est destructrice, et rend la compression non-conservative.

Le reste des traitements sur le macro-bloc courant correspondent à la production du *bitstream* (branche de gauche dans la figure 5.5), et la reconstruction de l'image encodée. Concernant la production du *bitstream*, la prédiction des coefficients INTRA (*IntraPredict*) est utilisée dans le cas des *frames* ou macro-blocs de type INTRA afin d'exploiter les corrélations spatiales de l'images de façon à réduire la variabilité des coefficients. L'étape finale consiste à appliquée une compression conservative (*Coding*), appelée parfois codage entropique, au coefficients issus des étapes précédentes. Cela permet d'encoder les coefficients avec le minimum de bits, en tenant compte des redondances que l'on retrouve dans les coefficients. Les algorithmes utilisés ici sont généralement le codage de Huffman ou le codage arithmétique. Le *bitstream* final contient également toutes les informations utiles pour décoder la *frame*, comme sa taille, son type, le type de chaque macro-bloc ou encore les vecteurs de mouvement pour les macro-blocs de type INTER.

La reconstruction de l'image se fait en appliquant les transformations inverses à la quantization, la transformée de Fourier discrète et l'estimation et compensation de mouvement. Cette étape permet de construire une *frame* qui correspond à la *frame* courante, mais en tenant compte des dégradation lié à la quantization. Cette *frame* reconstruite est donc identique à celle qui sera décodée par un décodeur. En effet, lors de l'encodage de la *frame* suivante, l'estimation et la compensation de mouvement se faire par rapport à la *frame* reconstruite et non la *frame* issue de la caméra vidéo, puisque un décodeur a connaissance de la première, et pas de la seconde.

Durées d'exécution

Après avoir présenté les différentes actions qui composent le modèle de l'application, ainsi que leur dépendances à l'aide du graphe hiérarchique, nous donnons les durées d'exécution moyennes et pire-cas. Ces durées d'exécution permettront la génération de l'application contrôlée, selon la technique présentée dans les chapitre 3 et 4. L'application que nous considérons ici possède huit niveaux de qualité $Q = \{\ 0,1,2,3,4,5,6,7,8\ \}$. Faute de posséder les outils d'analyse nécessaires, les durées d'exécution moyennes comme les durées d'exécution pire-cas ont été calculées à partir de traces d'exécution.

Les figures 5.6 et 5.7 donne les durées d'exécution moyennes et pire-cas, pour toutes les actions et tous les niveaux de qualité. Il est intéressant de noter que seules les actions

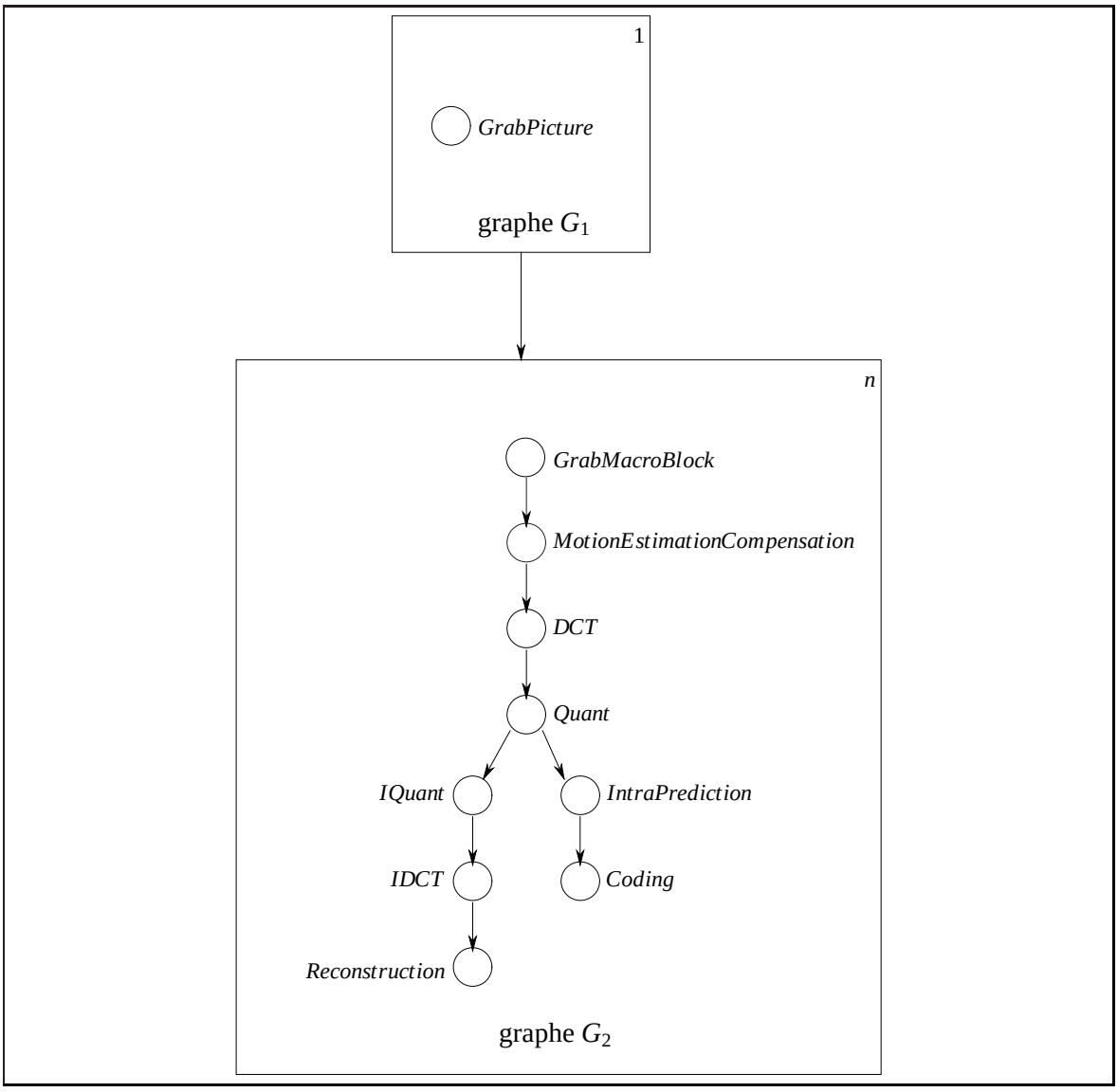


FIG. 5.5: Graphe hiérarchique de l'application d'encodage vidéo.

MotionEstimationCompensation et *DCT* sont “contrôlables”. En effet, les actions autres que *MotionEstimationCompensation* et *DCT* ne dépendent pas du paramètre de qualité q, ce qui se traduit des durées d’exécution indépendantes du niveau de qualité pour ces actions.

action / niveau de qualité q	0	1	2	3	4	5	6	7
GrabPicture	17000	17000	17000	17000	17000	17000	17000	17000
GrabMacroBlock	3.2	3.2	3.2	3.2	3.2	3.2	3.2	3.2
MotionEstimationCompensation	0.215	20	40	90	119	130	150	180
DCT	0.500	15	15	15	15	15	15	15
Quant	12	12	12	12	12	12	12	12
IntraPrediction	3.2	3.2	3.2	3.2	3.2	3.2	3.2	3.2
Coding	5	5	5	5	5	5	5	5
IQuant	4	4	4	4	4	4	4	4
IDCT	12	12	12	12	12	12	12	12
Reconstruction	9.5	9.5	9.5	9.5	9.5	9.5	9.5	9.5

FIG. 5.6: Durées d’exécution moyennes des actions en millier de cycles processeur.

action / niveau de qualité q	0	1	2	3	4	5	6	7
GrabPicture	40000	40000	40000	40000	40000	40000	40000	40000
GrabMacroBlock	5	5	5	5	5	5	5	5
MotionEstimationCompensation	1	100	200	400	500	1500	1500	2000
DCT	1	50	50	50	50	50	50	50
Quant	1	25	25	25	25	25	25	25
IntraPrediction	8	8	8	8	8	8	8	8
Coding	1	100	100	100	100	100	100	100
IQuant	10	10	10	10	10	10	10	10
IDCT	50	50	50	50	50	50	50	50
Reconstruction	19	19	19	19	19	19	19	19

FIG. 5.7: Durées d’exécution pire-cas des actions en millier de cycles processeur.

5.2 Comparaison entre une exécution contrôlée et une exécution à une qualité constante

Nous comparons ici l’application contrôlée par rapport à l’application dans laquelle nous avons fixé un niveau de qualité constant. Ceci correspond à la pratique industrielle courante, dans laquelle le seul contrôle qui est fait par rapport aux contraintes de temps est le *frameskip*. Le niveau de qualité est choisi en fonction de l’échéance globale, de façon à obtenir un compromis entre le taux de *frameskip* et la qualité de l’encodage. Dans les deux cas, la politique d’ordonnancement est statique, basée

l'ordonnancement

$\alpha_0 = GrabPicture\{GrabMacroBlockMotionEstimationCompensationDCT\,QuantIntraPredictionCodingIQuantIDC$

.

Nous avons mesuré la durée d'exécution de l'encodeur sur un flux vidéo de plus de 500 frames, composé de dix séquences vidéo différentes. Dans la figure 5.9, l'abscisse donne l'indice de la *frame* dans le flux vidéo, et l'ordonnée la durée de l'encodage de la *frame* correspondante. Le tracé en verte correspond à l'application contrôlée, pour laquelle il n'est pas nécessaire d'avoir une taille de tampon mémoire d'entrée de plus de 1. Le tracé rouge correspond à l'application exécutée intégralement au niveau de qualité 3, pour laquelle nous avons choisi de fixer la taille du tampon mémoire à 2, ce qui permet de réduire la taux de *frameskip*.

Nous remarquons qu'il existe neuf sauts communs aux deux tracés. Ceux-ci correspondent aux changements de plan qui existent entre les différentes séquences vidéo qui composent le flux vidéo d'entrée. La durée de l'encodage d'une *frame* qui correspond à un changement de plan, c'est-à-dire d'une *frame* de type INTRA, est en général très inférieure à celle de l'encodage d'une *frame* de type INTER. De plus, dans le cas d'une *frame* de type INTRA, les choix de contrôle sont très limités puisque la phase d'estimation et de compensation de mouvement n'existe plus. Ceci explique les neuf sauts communs aux deux tracés.

Nous remarquons également que, dans le cas de l'application exécutée au niveau de qualité constant, il existe deux séries de sauts qui ne sont pas présents dans l'application contrôlée. Ceux-ci correspondent à l'activation du *frameskip* lorsque le tampon mémoire est plein, ce qui conduit à une durée d'encodage quasi nul pour les *frames* concernées. Le *frameskip* correspond à une réduction brutale de la qualité. En effet, ne pas encoder pas une *frame* réduit la fluidité de la vidéo. Le *frameskip* tend également à diminuer la corrélation entre les *frames*, ce qui a pour effet d'augmenter la durée d'encodage. Avec l'application contrôlée, il n'y a pas de *frameskip*. En effet, le contrôleur évite la surcharge du système en réduisant progressivement le niveau de qualité durant l'encodage des *frames* dont les traitements sont les plus lourds. Le contrôleur assure donc le respect des échéances, ce qui évite l'utilisation d'un mécanisme de *frameskip* et/ou d'un tampon mémoire d'entrée. L'utilisation d'un niveau de qualité constant conduit également à une sous utilisation de la plate-forme lorsque les traitements des *frames* sont moins lourds. En augmentant le niveau de qualité, le contrôleur tente d'améliorer la qualité d'encodage de ces mêmes *frames*, ce qui conduit à une meilleure utilisation de la plate-forme.

L'observation des tracés nous conduit à penser que l'insertion d'un contrôleur dans l'application est un réel avantage. En effet, grâce à celui-ci nous obtenons une meilleure exploitation des ressources de calcul, que ce soit en cas de sous-charge ou de sur-charge du système. Ceci est obtenu par une adaptation progressive des niveaux de qualité, ce qui évite les fluctuations trop brutales de la qualité de la vidéo encodée. En considérant les échéances comme strictes, le contrôleur assure que l'application tourne exactement à la vitesse de la caméra, ce qui évite l'utilisation d'un tampon mémoire en entré ainsi qu'un mécanisme de *frameskip*. Or les *frame* prennent une place très importante en mémoire, ce qui devient problématique dans un univers fortement contraint tel que celui de l'embarqué. Pour finir, l'obtention d'une utilisation de presque 100 % du budget de temps est un résultat particulièrement

intéressant puisque les contraintes temps-réel sont considérées comme strictes. Il s'agit donc, dans une certaine mesure, d'une réconciliation entre le monde du temps-réel critique et le monde du temps-réel souple. Notons que ces bonnes performances s'expliquent par le fait que l'ordonnancement α_0 présente des valeurs δ^{max} faibles pour les niveaux de qualité qui sont mis en jeux. La figure 5.8 donne les valeurs de $\delta^{max}(\alpha_0, q)$ pour les différents niveaux de qualité.

q	0	1	2	3	4	5	6	7
$\delta^{max}(\alpha_0, q)$	54.3	148.5	189.3	339.3	410.3	1399.3	1379.3	1849.3
$\delta^{max}(\alpha_0, q)$ (ms) (ms)	0.00482	0.0132	0.01682	0.0301	0.03647	0.12438	0.12260	0.16438

FIG. 5.8: Valeurs $\delta^{max}(\alpha_0, q)$ pour tous les niveaux de qualité q en milier de cycles et en ms.

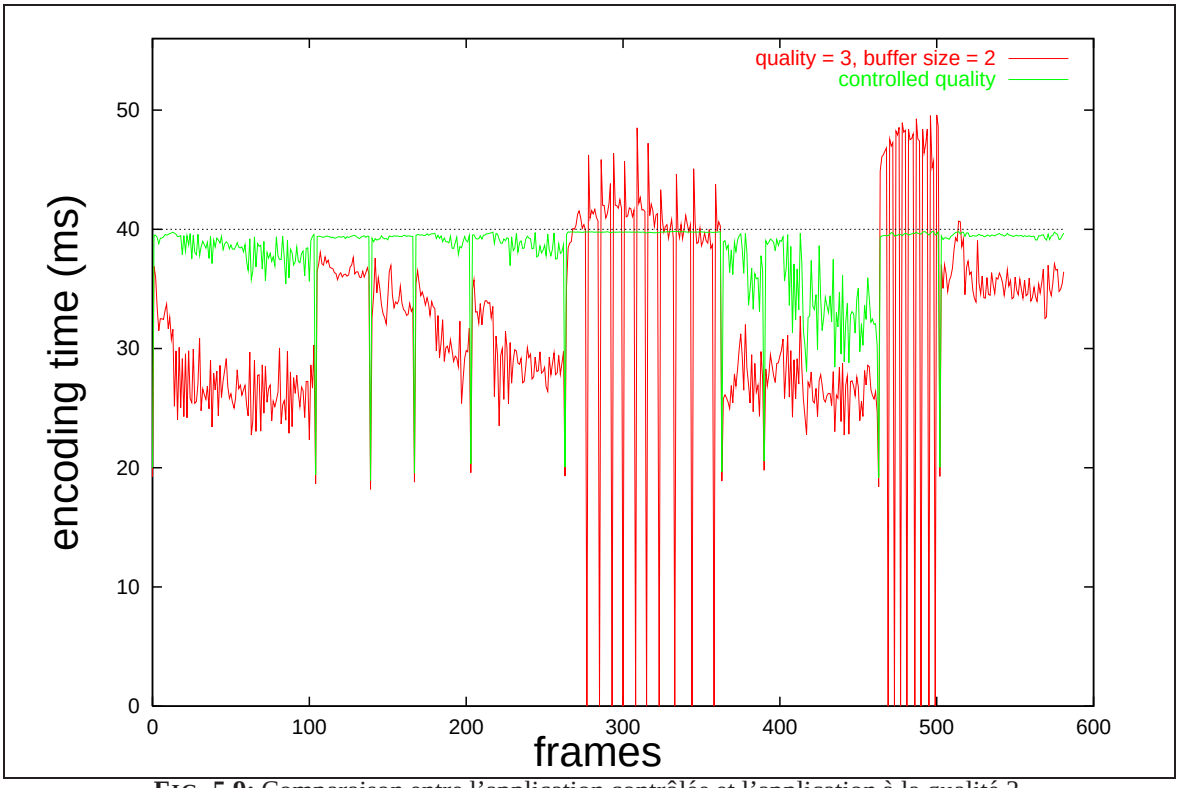


FIG. 5.9: Comparaison entre l'application contrôlée et l'application à la qualité 3.

5.3 Optimisation de la politique d'ordonnancement

Dans la section 3.4.2 du chapitre 3, nous avons vu comment optimiser la politique d'ordonnancement par rapport à la politique de gestion de qualité mixte. Dans la section 3.5 du même chapitre,

nous avons vu que, pour une politique d'ordonnancement statique basée sur un ordonnancement α_0, il existe un lien entre les valeur $\delta^{max}(\alpha_0, q)$, $q \in Q$, et l'utilisation du budget de temps. Plus précisément, nous avons démontré que sous certaines conditions, l'utilisation du budget de temps est d'au moins $(D - \delta^{max}(\alpha_0, q))/D$.

Nous avons comparé l'exécution de l'application contrôlée pour les politiques d'ordonnancement statiques basées sur les ordonnancements $\alpha_0 = GrabPicture\{\alpha_1\}^n\{\alpha_2\}^n$ et $\alpha'_0 = GrabPicture\{\alpha_1\alpha_2\}^n$, où α_1 correspond à l'ordonnancement de la branche de droite du graphe G_2 du modèle de l'application (voir la figure 5.5), c'est-à-dire :

$$\alpha_1 = GrabMacroBlockMotionEstimationCompensationDCTQuantIntraPredictCoding$$

et α_2 correspond à l'ordonnancement des actions restante, c'est-à-dire :

$$\alpha_2 = IQuantIDCTReconstruction.$$

Les valeurs $\delta^{max}(\alpha_0, q)$ et $\delta^{max}(\alpha'_0, q)$ sont données dans la figure 5.10. La politique de gestion de qualité utilisée sera la politique mixte.

q	0	1	2	3	4	5	6	7
$\delta^{max}(\alpha_0, q)$	86670.8	86765	86805.8	86955.8	87026.8	88015.8	87995.8	88465.8
$\delta^{max}(\alpha_0, q)$ (ms)	7.70407	7.71244	7.71607	7.72940	7.73571	7.82362	7.82184	7.86362

FIG. 5.10: Valeurs $\delta^{max}(\alpha'_0, q)$ pour tous les niveaux de qualité q, en millier de cycles et en ms.

Comme nous l'avons fait dans le paragraphe précédent, nous avons mesuré la durée d'exécution de l'encodeur pour le même flux vidéo d'entrée (figure 5.11). Nous remarquons que l'utilisation du budget de temps est de loin supérieur avec la politique d'ordonnancement statique basée sur α_0, qu'avec une politique d'ordonnancement statique basée sur α'_0. Nous avons mesuré une différence entre le durée d'encodage obtenue avec l'utilisation de α_0 et de α'_0 d'environ 7 ms en moyenne. Cette valeur correspond à la différence entre les valeurs δ^{max} obtenues en moyenne pour l'ordonnancement α_0 et α'_0, ce qui correspond bien à ce que nous attendions.

Les résultats expérimentaux montrent que l'optimisation de la politique d'ordonnancement est un facteur essentiel pour assurer une bonne utilisation de la plate-forme. De plus, la pertinence de notre critère d'optimisation, c'est-à-dire minimiser les valeurs δ^{max}, est confirmé dans la mesure où les traces d'exécution ont montré qu'il existe un lien direct entre l'utilisation de la plate-forme et la valeur les valeurs δ^{max} correspondant à la politique d'ordonnancement statique.

5.4 Comparaison des politiques de gestion de qualité

Nous comparons ici différentes exécution de l'application contrôlée pour différentes politiques de gestion de qualité. Nous présentons les résultats obtenus avec la politique de gestion de qualité sûre, moyenne et mixte, présentées respectivement dans les section 3.2.1, 3.3.2 et 3.4.1 du chapitre 3.

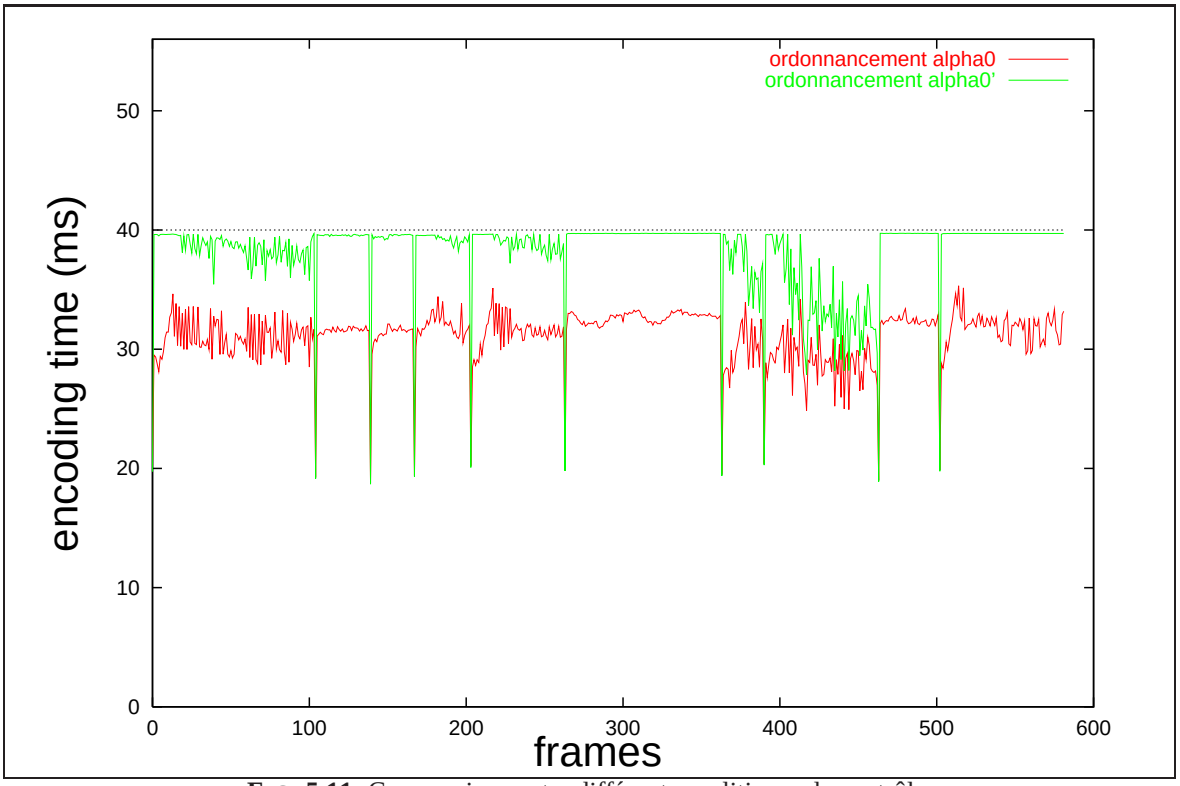


FIG. 5.11: Comparaison entre différentes politiques de contrôle.

Choix de la politique d'ordonnancement considéré

La politique d'ordonnancement choisie est encore statique, basée sur l'ordonnancement suivant :

$$\alpha_0 = GrabPicture\{\alpha_1\}^n\{\alpha_2\}^n,$$

où α_1 correspond à l'ordonnancement de la branche de droite du graphe G_2 du modèle de l'application (voir la figure 5.5), c'est-à-dire :

$$\alpha_1 = GrabMacroBlockMotionEstimationCompensationDCTQuantIntraPredictCoding$$

et α_2 correspond à l'ordonnancement des actions restante, c'est-à-dire :

$$\alpha_2 = IQuantIDCTReconstruction.$$

Puisque c'est l'action *Coding* qui produit le *bitstream*, l'ordonnancement α_0 est intéressant lorsqu'on cherche à minimiser la latence entre l'arrivée de la *frame* et la production du *bitstream*. Un tel ordonnancement est d'ailleurs obtenu par la politique d'ordonnancement EDF, lorsque l'échéance des actions de α_1 est plus petite que celle des actions de α_2.

Comparaison sur un diagramme des vitesses

Nous avons exécuté l'application contrôlée pour les politiques de gestion de qualité sûre, simple et mixte. La séquence vidéo d'entrée est la même que celle qui a été utilisée dans le paragraphe précédent. La figure 5.12 donne le diagramme des vitesses pour une *frame* particulière de la séquence d'entrée. Les exécutions de l'application pour la politique de contrôle sûre, simple et mixte sont représentées respectivement par les tracés rouge, bleu et vert sur le diagramme de la figure 5.12.

Dans le chapitre 3, nous avons vu que les politiques de gestion de qualité sûre, simple et mixte permettent toutes de respecter les échéances, ainsi que d'obtenir une utilisation élevée du budget de temps. Nous voyons en effet sur la figure 5.12 que l'utilisation du budget de temps est similaire pour les trois politique de gestion de qualité.

Concernant la régularité des niveaux de qualité choisis par le contrôleur, la différence est plus importante. Le diagramme des vitesses est une représentation intuitive pour étudier la régularité des niveaux de qualité. En effet, nous avons vu dans la section 3.5 que les niveaux de qualité correspondent à des vitesses constantes dans le diagramme des vitesses, lorsqu'on utilise une politique d'ordonnancement statique et que la fonction d'échéance est constante. Ainsi, si les durées d'exécution réelles sont proches des durées d'exécution moyennes, une affectations de qualité régulière devrait correspondre à un tracé presque rectiligne dans le diagrammes des vitesses. C'est d'ailleurs ce que nous constatons dans les expériences que nous avons mené, et en particulier dans celle qui est représentée sur le diagramme de la figure 5.12. En effet, les niveaux de qualité obtenus en utilisant la politique mixte sont presque constants, ce qui se traduit par un tracé presque rectiligne dans le diagramme. Les discontinuités de vitesse A et B (figure 5.12), présentent respectivement dans le cas de la politique de gestion de qualité sûre et simple, correspondent à l'activation du niveau de qualité minimal. Ces deux politiques sont donc trop optimistes au début de l'exécution, ce qui conduit à une réduction de qualité pour assurer le respect de l'échéance.

Il est évident que la politique de gestion de qualité sûre engendre une moins bonne régularité des niveaux de qualité que la politique simple et mixte, puisqu'elle ne tient pas compte des durées d'exécution moyennes, ce qui est confirmé par le diagramme. De plus, la politique de gestion de qualité sûre est basée sur la fonction de durée d'exécution sûre C^{sf}, avec laquelle la prochaine action à exécuter est comptées au niveau de qualité courant q, et les actions qui suivent sont comptées au niveau de qualité minimal. Ceci explique que la politique de gestion de qualité sûre soit trop optimiste dans ses choix de niveau de qualité pour le premières actions l'ordonnancement α_0.

La différence entre la politique de contrôle simple et mixte tient au fait que la politique simple anticipe mal l'impact de la sûreté sur les choix des niveaux de qualité. En effet, l'ordonnancement α_0 sur lequel est basé la politique d'ordonnancement est tel que les valeurs $\delta^{max}(\alpha_0, q)$ sont élevées (voir la figure 5.12). Cela signifie qu'il existe des suffixes ${\alpha_0}^k$ de α_0, c'est-à-dire des points de contrôle dans l'application, pour lesquels le comportement pire-cas prédomine fortement sur le comportement, c'est-à-dire la quantité $C^{sf}({\alpha_0}^k, q) - C^{av}({\alpha_0}^k, q) = \delta({\alpha_0}^k, q)$ est importante. Puisque la politique de gestion de qualité simple fait ses choix sans anticiper un tel phénomène, celle-ci se révèle trop optimiste pour les premières actions de α_0, ce qui est sur les résultats expérimentaux. En effet, sur le diagramme des vitesses de la figure 5.12, nous constatons que, même si la politique de gestion de qualité simple est moins optimiste que la politique de gestion de qualité sûre, elle conduit néanmoins à un

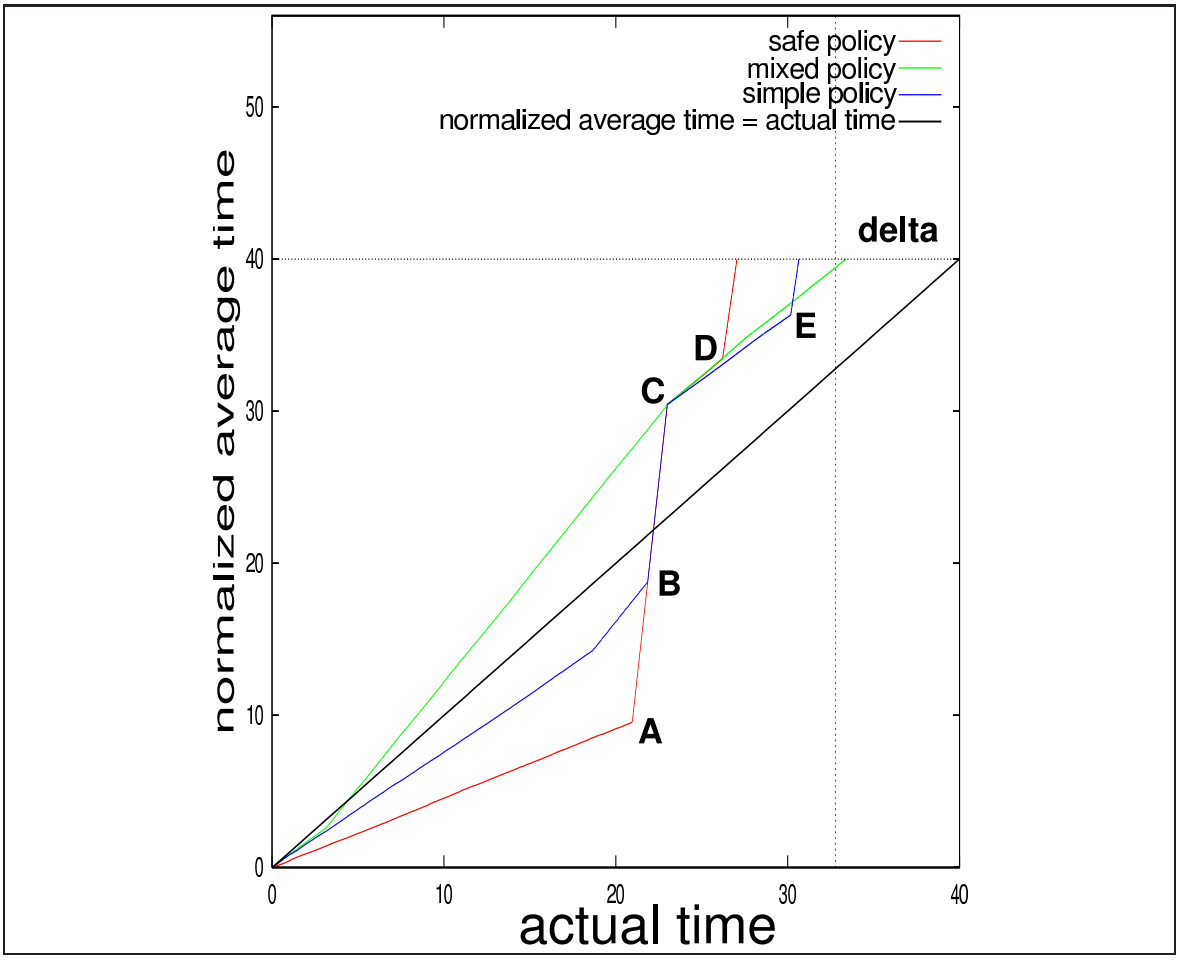


FIG. 5.12: Comparaison entre différentes politiques de contrôle.

tracé qui présente des discontinuité de vitesse, qui correspond à une affectation de qualité irrégulière.

Le point de discontinuité C, commun aux tracés de la politique sûre et simple, correspond au point de contrôle de l'application sur lequel il reste à exécuter le suffixe α_2 de α_0. Or les actions de α_2 ne sont pas "contrôlables", dans le sens où leurs durées d'exécution sont indépendantes du niveau de qualité. En effet, ces actions correspondent à des morceaux de code qui ne dépend pas du niveau de qualité choisi par le contrôleur.

Pour finir, les discontinuité D et E, présentent respectivement sur les tracés de la politique de contrôle sûre (tracé rouge) et de la politique de contrôle simple (tracé bleu), sont des effets de bords des discontinuités A et B. En effet, l'activation de la qualité minimale pour la première partie de α_0, c'est-à-dire pour les actions de α_1, induit des durées d'exécution très courtes pour les actions de α_2 qui correspondent au traitement des même macro-blocs. Ceci s'explique par le fait que les actions de α_2 sont des opérations inverses de certaines actions de α_1. De ce fait, il existe un lien entre les durées d'exécution des actions de α_1 et celles de α_2.

Les résultats expérimentaux confirment l'intérêt de la politique de contrôle mixte par rapport à la politique de contrôle simple et la politique de contrôle sûre. Si les deux dernières assurent le respect des échéances et une bonne utilisation de la plate-forme pour une politique d'ordonnancement donnée, elle ne conduisent pas à une bonne régularité des niveaux de qualité choisi par le contrôleur. En effet, alors que la politique simple et sûre conduisent à l'activation du niveau de qualité minimal, la politique de contrôle mixte permet conduit à une affectation de qualité presque constante. Ceci améliore sensiblement la qualité d'image finale.

CHAPITRE 6

Conclusion

Nous avons proposé une méthode de contrôle grain fin qui garantit le respect des contraintes temps-réel d'une application, tout en faisant une utilisation optimale des ressources de calcul. Ainsi, elle ne présente pas les limitations classiques des approches temps-réel critique dans lesquelles le respect des contraintes temps-réel conduit inévitablement à une faible utilisation de la puissance de calcul.

Notre technique de contrôle repose sur des bases théoriques solides. Nous avons donné un cadre général de construction de contrôleurs et de politiques de contrôle. Nous avons déterminé la classe des politiques de contrôle sûres, c'est-à-dire qui garantissent le respect des contraintes temps-réel sous l'hypothèse raisonnable que l'application est ordonnançable pour le niveau de qualité minimal et les durées d'exécution pire-cas. Nous avons expliqué pourquoi la politique de contrôle retenue — appelée politique de contrôle mixte — permet une bonne régularité des niveaux de qualité choisis par le contrôleur. L'étude théorique a également permis de quantifier l'impact du comportement pire-cas sur l'utilisation du processeur dans le cas de la politique de contrôle mixte. Ainsi, étant donné un modèle de l'application et des contraintes d'échéance, nous sommes en mesure de donner une estimation de l'utilisation du processeur. De ce fait, il est possible de prévoir si le contrôleur sera efficace, pour une application et une plate-forme données. Dans le cas contraire, il convient de revoir le choix des niveaux de qualité dans l'application et/ou le calcul des durées d'exécution pire-cas. En dernier recours, il est envisageable de relâcher le respect de certaines échéances, si l'application le permet.

L'intérêt de l'approche repose également sur une implémentation efficace qui respecte les critères

suivants.

Généricité. L'implémentation du contrôleur se devait d'être la plus générique possible quant au contexte d'exécution. Nous voulions en effet qu'il soit possible de changer la valeur des échéances ou le nombre d'itérations entre deux exécutions de l'application, ou même entre deux cycles de calcul différents, sans avoir à recompiler l'application. Dans le domaine de l'encodage vidéo, nous assurons de la sorte l'indépendance de l'implémentation vis à vis de la taille d'image ou encore du nombre d'images par seconde (*framerate*) du flux vidéo d'entrée. Il s'agit donc d'une propriété essentielle grâce à laquelle il est possible de générer une seule application pour tous les contextes d'exécution possibles.

Sur-coût faible. Maximiser l'utilisation du processeur a pour but de maximiser les niveaux de qualité et, donc, la qualité de service offerte par l'application. Si une part trop importante du budget de temps est accordée à l'exécution du contrôleur lui-même, la qualité de service sera inévitablement dégradée, aussi pertinents puissent être les choix du contrôleur. Il est donc nécessaire d'avoir une implémentation particulièrement légère du contrôleur. C'est un aspect critique au niveau de l'implémentation puisque le contrôle de l'application est réalisé à une granularité très fine. Mais c'est aussi une réelle difficulté. Les résultats expérimentaux ont montré qu'une telle implémentation est possible, puisque le sur-coût lié à l'appel au contrôleur est de moins de moins de 2 % pour l'application d'encodage vidéo, que ce soit en terme de consommation mémoire, durée d'exécution et taille de code généré.

Les résultats expérimentaux menés sur une application d'encodage vidéo confirment l'intérêt de l'approche. Le comportement de l'application contrôlée est en effet bien meilleur que celui d'une application s'exécutant à niveau de qualité constant — ce qui correspond à la pratique industrielle courante. Il en résulte une meilleure qualité de service dans le cas contrôlé, du fait d'une diminution progressive de la qualité d'image dans les situations de surcharge du processeur. De plus, l'application contrôlée ne nécessite pas de tampon mémoire d'entrée, autorisant une économie substantielle sur la quantité de mémoire nécessaire à son exécution. L'intérêt des politiques de gestion de qualité et d'ordonnancement proposées, notamment au niveau de la régularité de la qualité et de l'utilisation du processeur, a également été montré par nos résultats expérimentaux, confirmant ainsi l'intérêt de notre cadre de travail théorique. De même, la mesure théorique de l'utilisation du processeur rejoint celle que nous avons pu mesurer au cours de nos expériences. Nous estimons que tous ces résultats sont particulièrement encourageants quant à la validité et l'intérêt de ce travail.

Perspectives

Nous avons identifié un certain nombre d'améliorations ou d'extensions possibles de l'approche actuelle. Certaines peuvent facilement s'intégrer dans la technique de contrôle grain fin actuelle, d'autres nécessitent certainement un travail exploratoire conséquant.

Granularité de contrôle. Nous menons actuellement des travaux au sujet de la granularité de contrôle. Il s'agit de réduire les appels au contrôleur afin de diminuer le sur-coût lié à l'exécution du contrôleur. Pour ce faire, nous identifions les états de l'application pour lesquels il n'est pas nécessaire d'appeler le contrôleur, dans la mesure où nous sommes certain que le niveau de qualité ne peut pas

changer. Ces états correspondent à des points du diagramme des vitesses suffisamment éloignés des frontières dans le découpage en régions.

Temps-réel souple. L'approche actuelle considère que toutes les échéances sont strictes, et ne profite pas de la présence éventuelle d'un tampon mémoire d'entrée. La prise en compte de contraintes souples permettrait d'augmenter le champ d'application de notre méthode.

Modèle d'entrée. Le modèle d'entrée de l'application est essentiellement un graphe de précédence paramétré par des niveaux de qualité, augmenté des durées d'exécution moyennes et pire-cas, ainsi que des échéances. Le modèle des durées d'exécution pourrait être plus riche en étant basé, par exemple, sur une distribution des durées d'exécution (modèle stochastique).

Le graphe de précédence est une représentation assez naturelle pour les applications multimédia de type flot de données, qui sont justement visées par notre approche. Cependant, il subsiste des efforts à faire pour rendre l'approche utilisable dans le monde industriel. Tout d'abord, le graphe de précédence ne capte pas la notion de conditionnelle, ce qui ne permet pas une modélisation à un niveau de granularité intéressant lorsque l'algorithme comporte des structure conditionnelle à "haut niveau". Ensuite, le modèle des niveaux de qualité entier est très pauvre, et ne rend pas compte des interactions parfois très complexes qui peuvent exister entre le niveau de précision de calculs effectués dans les différentes parties l'application, et leur influence sur la qualité de service finale. Par exemple, toutes les zones d'une image ne nécessite pas forcément le même niveau de qualité d'encodage. De même, la répartition du budget de temps entre les différentes partie de l'application ne doit pas nécessairement être faite de manière uniforme, et peut dépendre des données. Pour finir, toutes les actions ne possèdent pas le même nombre de niveaux de qualité.

A l'heure actuelle, ces aspects doivent être traités à la main par des choix pertinents quant à la signification des niveaux de qualité dans le code applicatif. Il serait intéressant de faire évoluer le modèle de qualité de telle sorte que ces aspect puissent être traités par l'outil et non par le programmeur de l'application.

Architecture parallèle. A des fin de simplification, nous avons pris l'hypothèse d'une architecture monoprocesseur. Or ce n'est pas forcément réaliste dans le contexte du traitement multimédia, où l'on s'oriente le plus souvent vers des plates-formes multiprocesseur. Il serait intéressant de voir comment notre approche peut être étendue dans ce cas.

Multitâche. Une autre hypothèse simplificatrice dans notre approche impose à l'application d'être seule à s'exécuter sur le processeur. En particulier, ni l'exécution d'un système opérationnel, ni celle d'autres tâches ne doit perturber l'exécution de notre application contrôlée.

Différentes pistes s'offrent concernant la prise en compte du multitâche. Une manière simple est de considérer les échéances non plus comme dates absolues mais comme des contraintes sur le budget de temps alloué, en nombre de cycles processeur par exemple. Dans ce contexte, notre approche permet de générer des tâches contrôlées qui assurent une utilisation optimale de leur budget de temps : non dépassement du budget et maximisation de son utilisation. La répartition du budget de temps entre les différentes tâches serait faite en fonction du nombre de tâches et de leurs contraintes temps-réel. Si une nouvelle tâche est admise dans le système, les tâches contrôlées sont capables de s'adapter immédiatement au nouveau budget de temps qui leur est alloué. Ainsi, notre approche peut

être vue comme une technique de génération de composants robustes et flexibles.

Apprentissage des durées d'exécution. Les techniques d'apprentissage pourraient nous permettent d'actualiser le modèle des durées d'exécution moyennes en fonction des données d'entrée, afin de mieux prévoir le comportement de l'application et donc de faire de choix de contrôle plus pertinents. Il serait même possible de déterminer les durées d'exécution moyennes — et même pire-cas dans un contexte temps-réel souple — uniquement par apprentissage, facilitant la génération du modèle de l'application.

Optimisation de la consommation d'énergie. Nous étudions également d'autres pistes comme celles du contrôle de la fréquence d'horloge du processeur dans le cadre de la réduction de la consommation d'énergie électrique. Le contrôleur pourrait agir sur les niveaux de qualité et/ou la fréquence du processeur. La minimisation de la consommation ferait alors partie des critères d'optimisation.

ANNEXE A

Preuves

CETTE première annexe est consacrée aux démontrations que nous avons jugées trop longues ou trop techniques, et qui auraient pu nuire à la lecture du document.

A.1 Complexité du problème d'ordonnancement d'un système paramétré

Nous donnons d'abord une définition du problème du BIN-PACKING. Une instance de ce problème est composée d'un ensemble fini d'objets O, d'une fonction $v : O \rightarrow \mathbb{N}$ donnant le volume de chaque objet, et d'un volume V qui représente le volume du sac-à-dos. Le but est alors de trouver un sous-ensemble d'objets $v(O')$ tel que son volume total soit borné par V et soit maximal.

DÉFINITION A.1 (problème du BIN-PACKING) *Soit l'instance (O, v, V) telle que :*

1. *$O = \{ o_1, \ldots, o_n \}$ est un ensemble fini d'objets.*
2. *$v : O \rightarrow \mathbb{N}$ est une fonction associant à tout objet o son volume $v(o)$.*
3. *$V \in \mathbb{N}$ représente le volume du sac-à-dos.*

Etant donné une telle instance, le problème du BIN-PACKING *consiste à trouver un sous-ensemble d'objets $O' \subseteq O$ tel que son volume total $v(O')$ soit inférieur ou égal au volume du sac V, c'est-à-dire :*

$$V \geq v(O') = \sum_{o \in O'} v(o)$$

et tel que ce volume soit maximal, c'est-à-dire que s'il existe un sous-ensemble d'objets $O'' \subseteq O$ de volume total $v(O'')$ inférieur ou égal à V, alors le volume total de O' est au moins aussi grand que celui de O'' :

$$V \geq v(O'') \Rightarrow v(O') \geq v(O'').$$

DÉFINITION A.2 (problème de décision associé au problème du BIN-PACKING) *Une instance du problème de décision associé au problème du* BIN-PACKING *est un quadruplet $(O = \{ o_1, \ldots, o_n \}, v, V, W)$ tel que (O, v, V) est une instance du problème du* BIN-PACKING *est $W \in \mathbb{N}$ un volume tel que $V \leq WV$. Le problème de décision associé au problème du* BIN-PACKING *consiste à déterminer s'il existe ou non un sous-ensemble d'objets O' tel que son volume total soit compris entre W et V, c'est-à-dire :*

$$V \geq v(O') \geq W.$$

Nous dirons que ce problème est le problème de décision associé au problème du BIN-PACKING*, ou problème* D-BIN-PACKING.

DÉFINITION A.3 (problème de décision associé au problème d'ordonnnancement 2.2) *Une instance du problème de décision associé au problème d'ordonnancement 2.2 est un quintuplet (G, C, D, Q, q_0) tel que $(G = (A, \prec), C, D, Q)$ est un système paramétré et $q_0 \in \mathbb{N}$ un entier positif. Le problème de décision associé au problème du* BIN-PACKING *consiste à déterminer s'il existe ou non un ordonnanncement (α, θ) du système paramétré (G, C, D, Q) tel que $\sum_{a \in A} \theta(a) \geq q_0$.*

Le problème de décision associé au problème du BIN-PACKING (D-BIN-PACKING) est NP-complet [cite ...]. Nous allons montrer que le problème D-BIN-PACKING peut se réduire dans le problème du décision associé au problème d'ordonnancement 2.2.

Preuve de la proposition 2.8 : Nous utilisons la réduction suivante ψ. Soit $I = (O = \{ o_1, \ldots, o_n \}, v, V, W)$ une instance du problème D-BIN-PACKING. Considérons l'instance $(G = (A, \prec), C, D, Q, q_0) = \psi(I)$ du problème de décision associé au problème d'ordonnancement de la section 2.2.1, définie par :

1. L'ensemble d'actions est identique à l'ensemble des objets O, c'est-à-dire $A = O$.
2. Le graphe de précédence n'a aucun arc, c'est-à-dire $\prec = \emptyset$.
3. L'ensemble des niveaux de qualité Q correspond à l'ensemble de tous les volumes des objets, c'est-à-dire $Q = \{ 0 \} \cup \{ v(o_i) \mid o_i \in O \}$.
4. La fonction de durée d'exécution C est définie par :

$$C(o_i, q) = \begin{cases} 0 \text{ si } q = 0 \\ v(o_i) \text{ si } 0 < q \leq v(o_i) \\ 2D \text{ si } q > v(o_i). \end{cases}$$

5. La fonction d'écheance est constante égale à V, c'est-à-dire $\forall o_i \in A \,.\, D(o_i) = V$.
6. Le niveau de qualité q_0 correspond au volume W, c'est-à-dire $q_0 = W$.

Affecter le niveau de qualité nul à l'action o_i dans le problème d'ordonnancement, c'est-à-dire $\theta(o_i) = 0$, correspond à ne pas choisir l'objet o_i dans le sous-ensemble d'objet O' dans le problème D-BIN-PACKING. Affecter le niveau de qualité $v(o_i)$ à o_i, c'est-à-dire $\theta(o_i) = v(o_i)$, correspond à choisir l'objet o_i. Les niveaux de qualité strictement supérieures à $v(o_i)$ ne peuvent être affectée à o_i car ils conduisent à un coût supérieur à D ($2D$). Il est possible d'affecter les niveaux de qualité compris strictement entre 0 et $v(o_i)$ à l'action o_i, mais ceux-ci ne sont pas intéréssants dans la mesure où l'on cherche à maximiser les niveaux de qualité, et qu'il ne coûte pas plus cher d'ordonnancer o_i à la qualité $v(o_i)$.

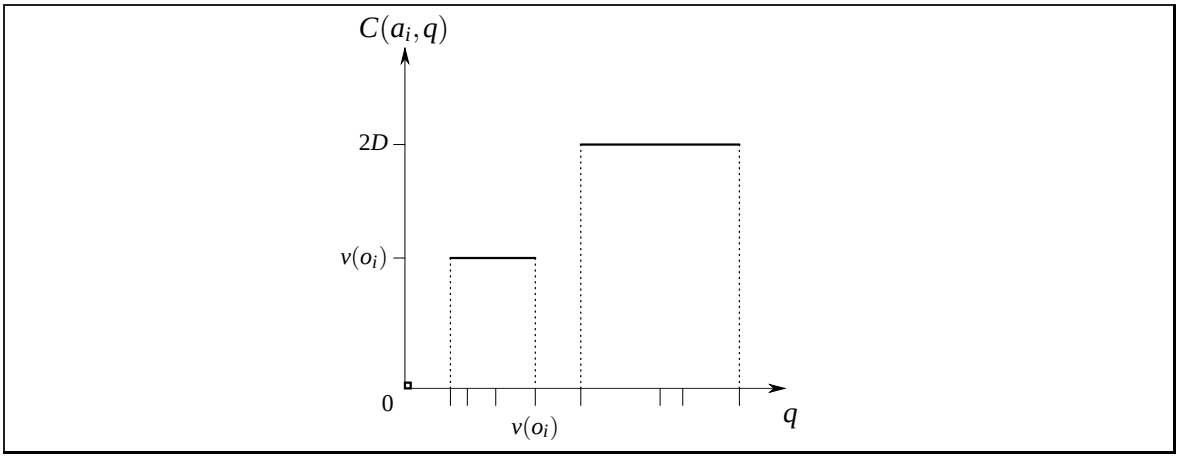


FIG. A.1: .

Tout d'abord, remarquons que la construction de cette instance du problème d'ordonnancement peut se faire en un temps polynomial en la taille de l'instance du problème D-BIN-PACKING. Nous allons démontrer dans la suite que (O, v, V, W) est une instance positive du problème D-BIN-PACKING si et seulement si $(G = (A, \prec), C, D, Q, q_0)$ est une instance positive du problème de décision associé au problème d'ordonnancement.

Soit α un ordonnancement quelconque de G. A tout sous-ensemble d'objets $O' \subseteq O$ nous associons l'affectation de qualité $\theta_{O'}$ définie par :

$$\theta_{O'}(a_i) = \begin{cases} v(o_i) \text{ si } o_i \in O' \\ 0 \text{ sinon.} \end{cases}$$

Nous avons l'équivalance suivante :

$$\begin{aligned} & V \geq v(O') \geq W \\ \Leftrightarrow \quad & V \geq \sum_{o \in O'} v(o) \geq W \\ \Leftrightarrow \quad & V \geq C(\alpha, \theta_{O'}) \geq W \\ \Leftrightarrow \quad & V \geq C(\alpha, \theta_{O'}) \wedge \sum_{a \in A} \theta_{O'}(a) \geq W \\ \Leftrightarrow \quad & D \geq C(\alpha, \theta_{O'}) \wedge \sum_{a \in A} \theta_{O'}(a) \geq q_0. \end{aligned}$$

Ainsi, s'il existe une réponse positive O' à l'instance (O, v, V, W) du problème D-BIN-PACKING, alors il existe une instance positive au problème de décision associé au problème d'ordonnancement.

Soit (α, θ) est une réponse positive au problème de décision associé au problème d'ordonnancement. Soit O'_θ le sous-ensemble d'objets associé à θ défini par :

$$O'_\theta = \{\ o_i \mid \theta(o_i) > 0\ \}.$$

Nous avons l'implication suivante :

$$\begin{aligned} & D \geq C(\alpha,\theta) \ \wedge \ \sum_{a\in A} \theta(a) \geq q_0 \\ \Rightarrow \quad & D \geq \sum_{o\in O'_\theta} v(o) = C(\alpha,\theta) \ \wedge \ \sum_{o\in O'_\theta} v(o) \geq \sum_{a\in A} \theta(a) \geq W \\ \Rightarrow \quad & V \geq v(O'_\theta) \geq W. \end{aligned}$$

Ainsi, s'il existe une réponse positive (α, θ) au problème de décision associé au problème d'ordonnancement, alors il existe une réponse O'_θ au problème D-BIN-PACKING.

Ainsi, nous avons montré que le problème D-BIN-PACKING se réduit polynomialement dans le problème de décision associè au problème d'ordonnancement. Puisque le problème du D-BIN-PACKING est NP-complet, nous en déduisons que le problème de décision associé au problème d'ordonnancement est aussi NP-complet, et donc que le probème d'ordonnancement est NP-difficile. □

A.2 Lemme 3.1

LEMME A.1 *Pour tout couple* (α, θ) *tel que* α *soit une séquence d'actions et* θ *une affectation de qualité définie sur* $\mathsf{ens}(\alpha)$*, nous avons :*

$$t_s^{sf}(\alpha,\theta) \leq t_s^{sf}(\alpha^2, q_{min}) - C^{wc}(\alpha(1),\theta).$$

Preuve du lemme A.1 : Nous avons :

$$t_s^{sf}(\alpha,\theta) \quad \leq \quad \mathbf{min}_{2\leq k\leq|\alpha|}\ t_s^{sf}(\alpha,\theta)(k) \tag{A.1}$$

où $t_s^{sf}(\alpha,\theta)(k) = D({}^k\alpha) - C^{sf}({}^k\alpha,\theta)$

$$= \quad D({}^k\alpha) - C^{wc}(\alpha(1),\theta) - C^{wc}(\alpha[2,k], q_{min}). \tag{A.2}$$

Pour tout $k \geq 2$, nous pouvons réécrire (A.2) de la façon suivante :

$$\begin{aligned} & D({}^{k-1}(\alpha^2)) - C^{wc}({}^{k-1}(\alpha^2), q_{min}) - C^{wc}(\alpha(1),\theta) \\ = \quad & t_s^{wc}(\alpha^2, q_{min})(k-1) - C^{wc}(\alpha(1),\theta) \\ = \quad & t_s^{sf}(\alpha^2, q_{min})(k-1) - C^{wc}(\alpha(1),\theta). \end{aligned} \tag{A.3}$$

D'après (A.1), (A.3), nous obtenons $t_s^{sf}(\alpha,\theta) \leq t_s^{sf}(\alpha^2, q_{min}) - C^{wc}(\alpha(1),\theta)$. □

Preuve du lemme 3.1 : La démontration est faite par induction sur les états atteignables.

• **initialisation :** état $(\varepsilon,\perp,0)$

Puisqu'il existe un ordonnancement α_0 de G tel que (α_0, q_{min}) est un ordonnancement réalisable de $PS(C^{wc})$, nous avons :

$$0 \leq t_s^{wc}(\alpha_0, q_{min}) = t_s^{sf}(\alpha_0, q_{min}).$$

Soit $\alpha_{q_{min}} = Best\_Sched^{sf}(\varepsilon, q_{min})$ l'ordonnancement prévu par l'ordonnanceur pour le niveau de qualité minimal à l'état $(\varepsilon,\perp,0)$. Il est facile de voir que les résultats de la proposition 2.10 s'appliquent aussi à t_s^{sf} et q_{min}. Puisque $\alpha_{q_{min}}$ est un ordonnancement EDF de G et que α_0 est un ordonnancement de G, par la proposition 2.10 nous obtenons :

$$0 \leq t_s^{sf}(\alpha_0, q_{min}) \leq t_s^{sf}(\alpha_{q_{min}}, q_{min}).$$

• **induction :** $(\alpha,\theta,t) \overset{(a,q_M)}{\rightarrow} (\alpha',\theta',t')$

Soit (α,θ,t) un état atteignable de $PS(C)||\Gamma$, et $(a,q_M) = \Gamma(\alpha,\theta,t)$. Soit (α',θ',t') un état $PS(C)||\Gamma$ tel que $(\alpha,\theta,t) \overset{(a,q_M)}{\rightarrow} (\alpha',\theta',t')$, $\alpha_q = Best\_Sched^{sf}(\alpha,q)$, and $\alpha'_q = Best\_Sched^{sf}(\alpha',q)$. Soit $q_M = \mathbf{max}\{q \mid t \leq t_s^{sf}(\alpha_q, q)\}$, et $a = \alpha_{q_M}(1)$.

Par hypothèse d'induction, nous avons $q_{min} \in \{q \mid t \leq t_s^{sf}(\alpha_q, q)\}$, c'est-à-dire $t \leq t_s^{sf}(\alpha_{q_{min}}, q_{min})$. Puisque q_M est le niveau de qualité choisi, nous avons aussi $t \leq t_s^{sf}(\alpha_{q_M}, q_M)$. Cette condition peut être réécrite, d'après le lemme A.1, de la façon suivante :

$$\begin{aligned} & t \leq t_s^{sf}(\alpha_{q_M}, q_M) \\ \Rightarrow\quad & t \leq t_s^{sf}({\alpha_{q_M}}^2, q_{min}) - C^{wc}(\alpha_{q_M}(1), q_M) \\ \Rightarrow\quad & t + C^{wc}(a, q_M) \leq t_s^{sf}({\alpha_{q_M}}^2, q_{min}). \end{aligned}$$

Puisque $C(a,q_M) \leq C^{wc}(a,q_M)$ et $t' = t + C(a,q_M)$, nous obtenons :

$$t' \leq t_s^{sf}({\alpha_{q_M}}^2, q_{min}).$$

Puisque $\alpha'_{q_{min}}$ est un ordonnancement EDF de $G/\alpha'_{q_{min}}$, et ${\alpha_{q_M}}^2$ est un ordonnancement du même graphe de précédence, d'après la proposition 2.10 appliquée à t_s^{sf} et q_{min}, nous avons $t_s^{sf}({\alpha_{q_M}}^2, q_{min}) \leq t_s^{sf}(\alpha'_{q_{min}}, q_{min})$, et nous obtenons $t' \leq t_s^{sf}(\alpha'_{q_{min}}, q_{min})$. Ceci démontre qu'à l'état successeur (α',θ',t'), nous avons encore $q_{min} \in \{q \mid t' \leq t_s^{sf}(\alpha'_q, q)\}$. Ainsi, à tout état atteignable (α,θ,t) of $PS(C)||\Gamma$, nous avons $q_{min} \in \{q \mid t \leq t_s^{sf}(\alpha_q, q)\}$. □

A.3 Proposition 3.19

LEMME A.2 *Soit $SPI(C)$ un système paramétré incertain et Γ^{mx} un contrôleur basé sur la politique de gestion de qualité mixte et une politique d'ordonnancement statique. Alors, si la fonction d'échéance D est constante et si $C = C^{av}$, l'ordonnancement (α_0,θ_0) calculé par le contrôleur est tel que $i \mapsto \theta_0(\alpha_0(i))$ est une fonction croissante.*

Preuve du lemme : Soit (α_1,θ_1,t_1) et (α_2,θ_2,t_2) deux états accessibles de $SPI(C)||\Gamma^{mx}$ tels que $(\alpha_1,\theta_1,t_1) \stackrel{(a,q)}{\rightarrow} (\alpha_2,\theta_2,t_2)$. Puisque Γ^{mx} est basé sur une politique d'ordonnancement statique, il existe i tel que $\alpha_1 =^i \alpha$ et $\alpha_2 =^{i+1} \alpha$. De plus, nous avons, pour tout $q \in Q$, $Best\_Sched^{mx}(\alpha_1,q) = \alpha^{i+1}$ et $Best\_Sched^{mx}(\alpha_2,q) = \alpha^{i+2}$. Nous allons montrer

$$q = \theta(\alpha(i+1)) \leq \theta(\alpha(i+2)). \tag{A.4}$$

Puisque q est le niveau de qualité choisi par le contrôleur à l'état (α_1,θ_1,t_1), soit il est admissible, soit c'est le niveau de qualité minimal q_{min}. Dans le second cas, nous obtenons immédiatement l'inégalité (A.4). Supposons que q est admissible à l'état (α,θ,t). Puisque D est constante, nous avons (voir section 3.4.2) :

$$\begin{aligned} t &\leq t_s^{mx}(\alpha^{i+1},q) = D - C^{av}(\alpha^{i+1},q) - \delta^{max}({\alpha_0}^{i+1},q) \\ t &\leq D - C^{av}(\alpha^{i+1},q) - \delta^{max}(\alpha^{i+1},q) \\ t &\leq D - C^{av}(\alpha(i+1),q) - C^{av}(\alpha^{i+2},q) - \delta^{max}(\alpha^{i+1},q) \end{aligned}$$

Puisque $C = C^{av}$ et que la politique d'ordonnancement est statique, basée sur α, nous avons

$$t_2 = t_1 + C^{av}(\alpha(i+1),q). \tag{A.5}$$

En utilisant les propriétés de la fonction δ^{max} (proposition 3.10 de la section 3.4.1), nous avons $\delta^{max}(\alpha^{i+1},q) = \delta^{max}(\alpha(i+1)\alpha^{i+2},q) \leq \delta^{max}(\alpha^{i+2},q)$. Ainsi, puisque D est constante nous obtenons nous réecrivons l'inégalité A.5 comme suit :

$$\begin{aligned} t &\leq D - C^{av}(\alpha(i+1),q) - C^{av}(\alpha^{i+2},q) - \delta^{max}(\alpha^{i+1},q) \\ t' &\leq D - C^{av}(\alpha^{i+2},q) - \delta^{max}(\alpha^{i+1},q) \\ t' &\leq D - C^{av}(\alpha^{i+2},q) - \delta^{max}(\alpha^{i+2},q) \\ t' &\leq D - t_s^{mx}(\alpha^{i+2},q). \end{aligned}$$

Nous avons donc montré que, si un niveau de qualité q est admissible sur un état de $SPI(C)$, alors il reste admissible par la suite. Ceci montre que la fonction $q \mapsto \theta(\alpha(i))$ est croissante. □

Preuve de la proposition 3.19 : D'après le lemme A.2, $i \mapsto \theta_0(\alpha(i))$ est une fonction décroissante. Par hypothèse, nous avons $\theta(\alpha(1)) = q < q_{max}$. Soit i le dernier indice tel que $\theta(\alpha(i)) = q$. Remarquons que i peut être égal à $|\alpha|$, auquel cas l'affectation de qualité θ est constante. Soit $t = C(^{i-1}\alpha,\theta) = C^{av}(^{i-1}\alpha,\theta)$ la valeur du temps-réel après avoir exécuté les actions $\alpha(1),\ldots,\alpha(i-1)$. L'état de $SPI(C)$ après avoir exécuté les actions $\alpha(1),\ldots,\alpha(i-1)$ est donc $(^{i-1}\alpha,q,t)$. Soit $Best\_Sched^{mx}(^{i-1}\alpha,q+1) = \alpha^i$ le prolongement prévu par l'ordonnanceur à l'état $(^{i-1}\alpha,q,t)$ pour le niveau de qualité $q+1$. Puisque le niveau de qualité choisi pour l'action $\alpha(i)$ est q, le niveau de qualité $q+1$ n'est pas admissible à l'état $(^{i-1}\alpha,q,t)$. Ainsi, puisque D est constante, nous avons :

$$\begin{aligned} & t_s^{mx}(\alpha^i,q+1) < t \\ \Rightarrow\ & D - C^{av}(\alpha^i,q+1) - \delta^{max}(\alpha^i,q+1) < C^{av}(^{i-1}\alpha,q) \\ \Rightarrow\ & D - C^{av}(\alpha^i,q+1) - \delta^{max}(\alpha^i,q+1) < C^{av}(^{i-1}\alpha,q). \end{aligned}$$

D'aprés la proposition 3.10, nous avons $\delta^{max}(\alpha^i, q+1) \leq \delta^{max}(\alpha, q+1)$. Nous en déduisons :

$$D - \delta^{max}(\alpha, q+1) < C^{av}({}^{i-1}\alpha, q) + C^{av}(\alpha^i, q+1). \tag{A.6}$$

En posant $a = \alpha(i)$, nous réécrivons la quantité $C^{av}({}^{i-1}\alpha, q) + C^{av}(\alpha^i, q+1)$ comme suit :

$$\begin{aligned} C^{av}({}^{i-1}\alpha, q) + C^{av}(\alpha^i, q+1) &= C^{av}({}^{i}\alpha, q) - C^{av}(\alpha(i), q) + C^{av}(\alpha^{i+1}, q+1) + C^{av}(\alpha(i), q+1) \\ &= C^{av}({}^{i}\alpha, q) + C^{av}(\alpha^{i+1}, q+1) + C^{av}(a, q+1) - C^{av}(a, q). \end{aligned}$$

Pour tout $j \leq i$, $\theta(\alpha(i)) = q$. De plus, puisque i est le dernier indice tel que $\theta(\alpha(i)) = q$, et grâce au lemme A.2, nous avons pour tout $j > i$ tel que $j \leq |\alpha|$ $\theta(\alpha(i)) \geq q+1$. Ainsi, $C^{av}({}^{i}\alpha, q) + C^{av}(\alpha^{i+1}, q+1) \leq C^{av}(\alpha, \theta)$. De plus, par définition de Δ, nous avons $C^{av}(a, q+1) - C^{av}(a, q) \leq \Delta$. Nous transformons ainsi l'inégalité A.6 de la façon suivante :

$$D - \delta^{max}(\alpha, q+1) - \Delta < C^{av}(\alpha, \theta) = C(\alpha, \theta). \qquad \square$$

TABLE DES FIGURES

BIBLIOGRAPHIE

[1] ISO/IEC 14496-10 H264.

[2] ISO/IEC 14496-2 MPEG-4.

[3] `http://iphome.hhi.de/suehring/tml/index.htm`.

[4] `http://www.chiariglione.org/mpeg/standards/mpeg-4/mpeg-4.htm`.

[5] `http://www.cplex.com`.

[6] `http://www.dashoptimization.com/`.

[7] `http://xirisc.deis.unibo.it/`.

[8] K. Altisen. Application de la synthèse de contrôleur à l'ordonnancement de systèmes temps-réel. thèse de doctorat. inpg., 2001.

[9] R. Alur and D. Dill. A theory of timed automata. *Theoritical Computer Science*, 1994.

[10] A. Atlas and A. Bestavros. Statistical rate monotonic scheduling. In *RTSS '98 : Proceedings of the IEEE Real-Time Systems Symposium*, page 123, Washington, DC, USA, 1998. IEEE Computer Society.

[11] N. C. Audsley, R. I. Davis, and A. Burns. Mechanisms for enhancing the flexibility and utility of hard real-time systems. In *Real-Time Systems Symposium*, pages 12–21. IEEE, 1994.

[12] V. Bertin, J.-M. Daveau, P. Guillaume, T. Lepley, D. Pilat, C. Richard, M. Santana, and T. Thery. FlexCC2 : An optimizing retargetable C compiler for DSP processors. In *EMSOFT*, pages 382–398, 2002.

[13] S. Brandt, G. Nutt, T. Berk, and J. Mankovich. A dynamic quality of service middleware agent for mediating application resource usage. In *RTSS '98 : Proceedings of the IEEE Real-Time Systems Symposium*, page 307, Washington, DC, USA, 1998. IEEE Computer Society.

[14] R. J. Bril, M. Gabrani, C. Hentschel, G. C. van Loo, and E. F. M. Steffens. QoS for consumer terminals and its support for product families. In *Proceedings of the International Conference on Media Futures*, 2001.

[15] A. Burns and A. J. Wellings. Criticality and utility in the next generation. *Real-Time Systems*, 3(4) :351–354, 1991.

[16] G. C. Buttazzo. *Hard Real-Time Computing Systems*. Kluwer Academic, 2000.

[17] J.-Y. Chung, J. W. S. Liu, and K.-J. Lin. Scheduling periodic jobs that allow imprecise results. *IEEE Trans. Comput.*, 39(9) :1156–1174, 1990.

[18] G. B. Dantzig and M. N. Thapa. *Linear Programming*. Springer.

[19] J.-M. Daveau, T. Thery, T. Lepley, and M. Santana. A retargetable register allocation framework for embedded processors. In *LCTES '04 : Proceedings of the 2004 ACM SIGPLAN/SIGBED conference on Languages, compilers, and tools for embedded systems*, pages 202–210, New York, NY, USA, 2004. ACM Press.

[20] R. I. Davis, K. W. Tindell, and A. Burns. Scheduling slack time in fixed priority preemptive systems. In *Proceeding of the IEEE Real-Time Systems Symposium*, pages 222–231, 1993.

[21] G. Franklin, D. Powell, and M. L. Workman. *Digital Control of Dynamic Systems*. Addison Wesley Publishing Co.

[22] P. Gopinath and A. Gupta. Applying compiler techniques to scheduling in real-time systems. In *IEEE Real-Time Systems Symposium*, pages 247–256, 1990.

[23] S. Graham and P. Kumar. The convergence of control, communication, and computation, 2003.

[24] D. Haban and K. G. Shin. Application of real-time monitoring to scheduling tasks with random execution times. *IEEE Trans. Softw. Eng.*, 16(12) :1374–1389, 1990.

[25] R. Heckmann, M. Langenbach, S. Thesing, and R. Wilhelm. The influence of processor architecture on the design and the results of WCET tools. *Proceedings of the IEEE*, 91(7) :1038–1054, 2003.

[26] D. Isovic, G. Fohler, and L. Steffens. Timing constraints of MPEG-2 decoding for high quality video : misconceptions and realistic assumptions.

[27] L. Khachiyan. A polynomial time algorithm in linear programming (english translation). *Soviet Mathematics Doklady*, 20 :191–194, 1979.

[28] G. Koren and D. Shasha. Skip-over : Algorithms and complexity for overloaded systems that allow skips. Technical Report TR1996-715, , 1996.

[29] G. Lafruit, L. Nachtergaele, K. Denolf, and J. Bormans. 3D computational graceful degradation. In *The 2000 IEEE International Symposium on Circuits and Systems*, volume 3, pages 547–550. IEEE, 2000.

[30] J. Lehoczky, L. Sha, and Y. Ding. The rate monotonic scheduling algorithm : Exact characterization and average case behavior. In *Real-Time Systems Symposium*, pages 201–209, 1989.

[31] J. Lehoczky, L. Sha, and J. K. Strosnider. An optimal algorithm for scheduling soft-aperiodic tasks fixed-priority preemptive systems. In *Proceedings of the IEEE Real-Time System Symposium*, pages 110–123, 1992.

[32] J. Lehoczky and S.Thuel. Algorithms for scheduling hard aperiodic tasks in fixed-priority systems using slack stealing. In *Proceedings of the IEEE Real-Time System Symposium*, 1994.

[33] C. L. Liu and J. W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM*, 20(1) :46–61, 1973.

[34] K. J. Liu, S. Natarajan, and J.-S. Liu. Imprecise results : Utilizing partial computations in real-time systems. In *Real-Time Systems Symposium*. IEEE, 1987.

[35] K. J. Liu, S. Natarajan, J.-S. Liu, and T. Krauskopf. Scheduling real-time, periodic jobs using imprecise results. In *Real-Time Systems Symposium*. IEEE, 1987.

[36] K. J. Liu, S. Natarajan, J.-S. Liu, and T. Krauskopf. Concord : A system of imprecise computations. In *Compsac*. IEEE, Oct. 1987.

[37] C. Lu, J. Stankovic, G. Tao, and S. Son. Feedback control real-time scheduling : Framework, modeling and algorithm. *special issue of RT Systems Journal on Control-Theoric Approach To Real-Time Computing*, 23(1/2) :85–88, 2002.

[38] M. Mattavelli, S. Brunetton, and D. Mlynek. Computational graceful degradation for video sequence decoding. In *ICIP '97 : Proceedings of the 1997 International Conference on Image Processing (ICIP '97) 3-Volume Set-Volume 1*, page 330, Washington, DC, USA, 1997. IEEE Computer Society.

[39] K. D. Nilsen and B. Rygg. Worst-case execution time analysis on modern processors. In *Workshop on Languages, Compilers, Tools for Real-Time Systems*, pages 20–30, 1995.

[40] L. Papalau, C. M. O. Pérez, and L. Steffens. In S. Goddard, editor, *Work-In-Progress Session of the 16th Euromicro Conference on Real-Time Systems*, pages 33–36, 2004.

[41] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *18th ACM Symposium on Operating Systems Principles, SOSP'01C*. IEEE, 2001.

[42] N. Report. The economic impacts of inadequate infrastructure for software testing. Technical report.

[43] F. Rovati, D. Pau, E. Piccinelli, L. Pezzoni, and J.-M. Bard. An innovative, high quality and search window independent motionestimation algorithm and architecture for mpeg-2 encoding. In *IEEE Transactions on Consumer Electronics*, volume 46, pages 697–705. IEEE, 2000.

[44] V. Sarkar. Determining average program execution times and their variance. In *SIGPLAN Conference on Programming Language Design and Implementation*, pages 298–312, 1989.

[45] G. M. Schuster, G. Melnikov, and A. K. Katsaggelos. A review of the minimum maximum criterion for optimal bit allocation among dependent quantizers. *IEEE Transactions on Multimedia*, 1(1) :3–17, 1999.

[46] J. A. Stankovic. VEST — A toolset for constructing and analyzing component based embedded systems. *Lecture Notes in Computer Science*, 2211 :390– ? ?, 2001.

[47] P. Westerink, R. Rajogopalan, and C. Gonzales. Two-pass MPEG-2 variable bit-rate encoding. 43(4), 1999.

[48] M. Woodbury. Analysis of the execution time of real-time tasks. In *RTSS*, pages 89–96, 1986.

[49] M. Woodbury and K. Shin. Evaluation of the probability of dynamic failure and processor utilization for real-time systems. In *RTSS*, 1988.

[50] C. C. Wüst, L. Steffens, R. J. Bril, and W. F. Verhaegh. Qos control strategies for high-quality video processing. In *Euromicro Conference on Real-Time Systems*, pages 3–12. IEEE, 2004.

INDEX