Ordonnancement sous contraintes (m,k)-firm et combinatoire des mots

Ning Jia, Emmanuel Hyon, Ye-Qiong Song

Ordonnancement sous contraintes (*m,k*)-firm et combinatoire des mots

Ning JIA, Emmanuel HYON, Ye-Qiong SONG

LORIA – INPL – UHP Nancy 1
Campus Scientifique - B.P. 239
54506 VANDOEUVRE-lès-NANCY, France

{jia, song}@loria.fr, ehyon@u.paris10.fr

Résumé

Dans ce papier, nous montrons une nouvelle méthode d'analyse de l'ordonnançabilité des ensembles de tâches sous contraintes (*m,k*)-firm en utilisant des propriétés des mots mécaniques (théorie des mots). Nous nous intéressons à l'ordonnancement sous contrainte de (*m,k*) pattern fixe. Dans un premier temps, nous montrons que les patterns introduits dans la littérature se caractérisent bien sous la forme de mots mécaniques. Les preuves d'ordonnançabilité en sont ainsi simplifiées. En identifiant les défauts de ces patterns, nous proposons ensuite une nouvelle technique basée sur la ligne cellulaire pour déterminer les (*m,k*) patterns des tâches. Les résultats expérimentaux montrent que cette nouvelle technique permet une amélioration de la région ordonnançable.

Mots clés : Ordonnancement temps réel, contrainte (*m,k*)-firm, mot mécanique

Abstract

In this paper, we show a new method for the schedulability analysis of the task sets under (*m,k*)-firm constraints using the properties of the mechanical words. We are only interested in the scheduling problem under fixed (*m,k*) patterns. First, we show that the patterns defined in the literature can be characterized in the form of the mechanical words with which the schedulability proofs are largely simplified. Then, by identifying the defaults of these patterns, we propose a new way, based on the cellular line, to determine the (*m,k*) patterns of the tasks. The experimental results showed that our approach achieves an improvement of the schedulable region.

Keywords: Real-time scheduling, (*m,k*)-firm constraint, Mechanical word

1. Introduction

Le déploiement sans cesse croissant des systèmes temps réel allant des applications embarquées critiques à celles sur l'internet souligne la nécessité d'assouplir des hypothèses classiques des modèles de tâches de l'ordonnancement temps réel dur et des contraintes strictes afin que la solution d'ordonnancement soit la plus robuste possible face aux aléas de ressources et des instants d'activation des tâches. Aujourd'hui, pour de nombreux systèmes temps réel, la qualité de service requise pour le bon fonctionnement s'exprime non plus par le strict respect des échéances de toutes les instances mais par une proportion d'instances respectant strictement leurs échéances temporelles, les instances restantes étant du temps réel souple. De tels systèmes sont par exemple des flux multimédias sur l'Internet (vidéo, voix par exemple). C'est à partir de cette caractérisation moyenne du respect des échéances que la notion de temps réel "*firm*" est définie [5]. Ainsi, une tâche est définie comme étant sous la contrainte (m,k)-firm si au moins m instances parmi k instances consécutives quelconques (avec $m < k$) respectent leurs échéances. Cette définition trouve sa justification dans le fait que supprimer une instance peut parfois améliorer la possibilité du respect des échéances des autres.

Une présentation des différents algorithmes d'ordonnancement sous contrainte (m,k)-firm apparaît dans [10]. Pour l'ordonnancement à priorité fixe, [9] propose un algorithme d'ordonnancement appelé ERM (Enhanced Rate Monotonic), qui classifie les instances de chaque tâche, selon un (m,k) pattern fixe, en deux ensembles : instances critiques, qui doivent vérifier toutes leurs échéances, et instances optionnelles dont les échéances peuvent être manquées. Ainsi, si toutes les instances critiques respectent leurs échéances, la contrainte (m,k)-firm est alors respectée [9]. Mais cet algorithme ne prend pas en compte la superposition (ou interférence) dans le temps des instances critiques des différentes tâches, ce qui limite la région ordonnançable (ou faisable). L'approche proposée dans [8] permet de mieux prendre en compte les interférences entre les tâches et ainsi d'améliorer l'ordonnançabilité de [9]. Mais à l'intérieur d'un (m,k) pattern, les instances critiques se concentrent toujours autour d'instants particuliers, ce qui réduit inévitablement la région ordonnançable.

Dans ce papier, nous allons tout d'abord nous intéresser à la classification des instances qui est un mot appelé (m,k) pattern dont les lettres décrivent l'état de l'instance (optionnelle ou critique). Nous allons aborder l'étude de ces patterns par la théorie des mots et en particulier des mots mécaniques (ou mots de Sturm). Nous allons montrer que les patterns de [9] et [8] sont des mots mécaniques et qu'un certain nombre de résultats de [9] découlent directement des propriétés de ces mots. Ensuite, afin de diminuer les points de concentration des instances critiques dans un pattern, nous proposons une technique de distribution des instances critiques dans un pattern basé sur des mots appelés *lignes cellulaires* [4]. Les expérimentations menées montrent que cette approche permet d'obtenir de meilleurs résultats.

Le papier s'organise comme suit. La section 2 présente les (m,k) patterns existants. Dans la section 3, nous étudions d'abord la relation entre les mots mécaniques et les patterns, puis montrons la simplification des preuves de l'ordonnançabilité grâce à des propriétés des mots

mécaniques. La section 4 présente notre approche basée sur les lignes cellulaires. Tandis que la section 5 donne les résultats expérimentaux avant de conclure le papier dans la section 6.

2. Préliminaires

Le cadre général de l'étude est le modèle MIQSS (*Multiple Input Queues Single Server*) dans lequel des flux de demandes, appelés tâches, vont être traités par une seule ressource appelée serveur. Pour ordonnancer les demandes de travail, appelées instances, la politique utilisée dans le serveur est la politique d'ordonnancement préemptif à priorité fixe (FPP) [6], c'est à dire que les priorités des tâches sont fixes au cours du temps et à chaque instant le serveur est alloué à l'instance la plus prioritaire.

Nous considérons un système avec n tâches périodiques indépendantes, $\Gamma = \{\tau_1,\ldots\tau_n\}$, ordonnées dans l'ordre décroissant de leur priorité (τ_i est plus prioritaire que τ_j si $i < j$). Nous supposons que toutes les tâches commencent en même temps (toutes les premières instances demandent du travail à l'instant 0). Les paramètres pour une tâche τ_i ($i \in \{1..n\}$) sont définis comme suit. La $j^{\text{ième}}$ instance de τ_i est notée $\tau_{i,j}$. Le temps d'exécution d'une instance sur la ressource est noté C_i. La période de la tâche : le temps qui sépare les arrivées de deux instances consécutives de la tâche τ_i se note T_i. Conformément à [6] la valeur de T_i détermine la priorité des tâches selon "*Rate Monotonic*" ainsi ($T_i<T_j$ <=> $i<j$). L'échéance relative : l'intervalle de temps autorisé entre la date d'arrivée d'une instance et sa date de fin d'exécution est noté D_i. On suppose que l'échéance est sur requête *i.e.* $T_i = D_i$. Les termes m_i et k_i représentent la contrainte (m,k)-firm de τ_i, qui signifie que, au moins, m_i instances parmi k_i instances consécutives quelconques de τ_i doivent avoir fini leurs exécutions avant leurs échéances. Ces m_i instances sont appelées *instances critiques*, tandis que les k_i-m_i autres sont appelées *instances optionnelles*. Toutes les instances critiques d'une tâche i, τ_i (avec i quelconque), ont la priorité de τ_i alors que l'ensemble des instances optionnelles de toutes les tâches est regroupé de manière à former un $(n+1)^{ème}$ flux qui se voit assigné de la plus faible priorité (la priorité $n+1$).

Pour garantir le respect de la contrainte (m,k)-firm dans le cas général, il suffit que n'importe lesquelles m parmi k instances consécutives quelconques respectent leur échéances ((m,k) *pattern dynamique*) [5]. Néanmoins, pour des applications particulières telles que la transmission des flux MPEG [11], il est intéressant de fixer le profil du respect selon un *(m,k) pattern fixe*. Une technique particulière de classification des instances peut alors se faire, pour chaque instance de chaque tâche, selon une variable binaire avec '1' pour désigner une instance critique et '0' une instance optionnelle. La détermination de l'ensemble des instances en critique et optionnelle forme alors un mot. Il a été prouvé dans [9] que le respect du (m,k) pattern fixe implique le respect de la contrainte (m,k)-firm mais l'inverse n'est pas vraie.

Définition 2.1 ((m,k) pattern). *Le (m,k) pattern de la tâche* τ_i*, noté* Π_i*, est une chaîne de caractères* $\Pi_i = \{\Pi_i(0)\Pi_i(1)...\Pi_i(k_i-1)\}$ *telle que* $\tau_{i,j}$ *est une instance critique si* $\Pi_i(j-1) = 1$*, et une instance optionnelle si* $\Pi_i(j-1) = 0$ *et telle que* $\sum_{j=0}^{k_i-1}\Pi_i(j) = m_i$.

Le (*m,k*) pattern est un mot fini et la qualification de l'ensemble des instances est un mot infini périodique résultant de la concaténation à l'infini du pattern fini.

2.1 Ordonnancement et (*m,k*) pattern

L'ordonnancement (*m,k*)-firm consiste à déterminer quelles sont les instances optionnelles et les instances critiques afin de trouver le placement des unes par rapport aux autres qui permette l'ordonnançabilité des instances critiques. Dans l'algorithme de Ramanathan [9], la classification d'instances en critiques et optionnelles se fait de la manière suivante :

Définition 2.2 ((*m,k*) pattern de Ramanathan). *Soit la tâche τ_i de paramètres m_i et k_i. La classification des instances critiques et optionnelles est donnée en fonction du mot binaire dont la lettre d'indice j vaut :*

$$\Pi_i(j) = \begin{cases} 1 \ si \ j = \left\lfloor \left\lceil j \cdot \frac{m_i}{k_i} \right\rceil \cdot \frac{k_i}{m_i} \right\rfloor \\ 0 \ \text{sinon} \end{cases} \qquad j = 0,1...k_i - 1 \tag{1}$$

Le mot déterminé selon cette formule est appelé (m,k) pattern de Ramanathan.

Exemple 2.3. *Soit un ensemble composé de deux tâches τ_1 et τ_2. La contrainte (m,k) de τ_1 est (2,3) et la contrainte (m,k) de τ_2 est (5,9). Selon la formule (1), $\Pi_1(0) = 1$ car $0 = \lfloor \lceil 0 \cdot 2/3 \rceil \cdot 3/2 \rfloor$; $\Pi_1(1) = 1$ car $1 = \lfloor \lceil 1 \cdot 2/3 \rceil \cdot 3/2 \rfloor$ et ainsi de suite. Le (m,k) pattern de τ_1 est alors 110 et celui de τ_2 est 110101010.*

La classification d'instances de [9] ne dépend que des valeurs de *m* et *k* sans prendre en compte la superposition des instances critiques entre des tâches différentes. On constate que les instances critiques des tâches auront tendances à être toutes placées au début de chaque (*m,k*) pattern, ce qui risque de réduire la région d'ordonnançabilité.

Pour remédier à ce problème, Quan [8] a proposé un algorithme heuristique (Il a été d'ailleurs prouvé dans [8] que le placement optimal est NP-complet) qui vise à améliorer l'ordonnançabilité des patterns de Ramanathan en permutant des sous mots du pattern de Ramanathan pour trouver un meilleur étalement des instances critiques. Pour ce faire, la notion d'*interférence d'exécution* est introduite afin d'estimer l'influence d'une tâche plus prioritaire sur une tâche moins prioritaire.

Définition 2.4 (Interférence d'exécution). *Soit deux tâches τ_h et τ_i (*avec *τ_h plus prioritaire que τ_i) et soit le* (*m,k*) *pattern de chaque tâche. L'interférence d'exécution de τ_h sur l'instance critique $\tau_{i,j}$, dénotée F_{ij}^h, est égale à la portion totale du temps d'exécution de toutes les instances critiques de la tâche τ_h dans l'intervalle $\left[(j-1)\cdot T_i, j \cdot T_i\right]$.*

Les interférences d'exécution de la tâche τ_h sur les instances critiques de τ_i peuvent prendre plusieurs valeurs différentes à cause du décalage entre les deux périodes. La plus grande interférence d'exécution subie par τ_i est notée $F_i^h = \max_j \left\{ F_{ij}^h \right\}$, avec $j = 1,2,..,\infty$. Intuitivement, réduire F_i^h permet d'accroître l'ordonnançabilité de τ_i.

Afin de réduire la plus grande interférence d'exécution et de savoir quand elle se produit, [8] définit la notion du *pire point de concentration* des instances critiques:

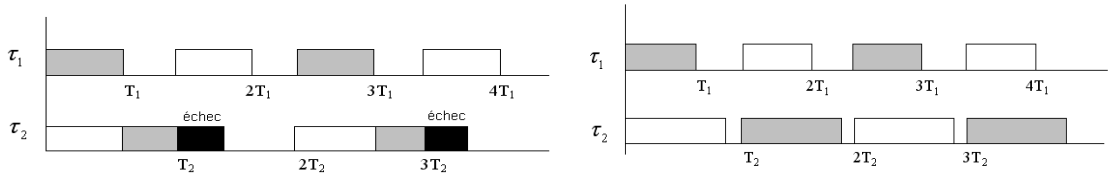
Définition 2.5 (Pire point de concentration des instances critiques). *Le pire point de concentration des instances critiques d'une tâche τ_i, appelé WCIP (worst-case interference point), est l'instant tel que le nombre d'instances critiques de la tâche τ_i est le plus grand, dans tout intervalle commençant à cet instant par rapport à tous les autres intervalles de même longueur.*

Lorsque le (*m,k*) pattern de la tâche τ_i est un pattern de Ramanathan, les WCIP se trouvent aux instants nk_iT_i, pour tout $n \in \mathbb{N}$, c'est à dire aux débuts des (*m,k*) patterns. Le principe de [8] consiste à transformer les patterns de Ramanathan, afin de décaler temporellement les instants de début de tous les (*m,k*) patterns et ainsi de décaler les WCIP entre les tâches. Ceci se fait en trouvant une valeur s_i ($0 \le s_i \le k_i$ *et* $s_i \in \mathbb{N}$), pour que le nouveau pattern, appelé pattern de Quan, soit calculé par :

$$\pi_{ij} = \begin{cases} 1 \ si \ j + s_i = \left\lfloor \left\lceil (j+s_i) \cdot \dfrac{m_i}{k_i} \right\rceil \cdot \dfrac{k_i}{m_i} \right\rfloor \\ 0 \ \text{sinon} \end{cases} \qquad j = 0,1 \ldots k_i - 1 \tag{2}$$

De cette manière, la plus grande interférence d'exécution d'une tâche plus prioritaire sur une autre moins prioritaire sera réduite. De plus, les temps de réponse des premières instances critiques seront moindres. Aussi le domaine de faisabilité est amélioré.

Considérons l'exemple suivant, illustré sur la figure 1. Soit un ensemble composé de deux tâches τ_1 et τ_2 dont les contraintes (*m,k*) sont (1,2). Si les (*m,k*) patterns sont les patterns de Ramanathan, alors les WCIP de ces deux tâches ont lieu au même moment, ce qui va rendre la tâche 2 non ordonnançable. En utilisant l'algorithme proposé dans [8], le (*m,k*) pattern de Quan de la tâche 2 est 01. Les dates d'arrivée des instances critiques sont décalées le plus possible ce qui permet l'ordonnançabilité.



(a) en prenant 10 comme le (*m,k*) pattern de τ_2, cet ensemble n'est pas ordonnançable.

(b) en prenant 01 comme le (*m,k*) pattern de τ_2, cet ensemble est ordonnançable.

Figure 1. Les patterns de Quan permettent le respect des échéances.

3. Mots et (*m,k*) patterns

On trouvera dans cette partie un bref survol des notions de la théorie des mots et plus spécifiquement des mots mécaniques. On s'attachera ensuite à mettre en lumière la relation entre mot mécanique et (*m,k*) pattern.

3.1 Notations préliminaires

Introduisons notations et définitions générales, relatives au formalisme des mots finis et infinis. La plupart d'entre elles proviennent de [7].

Définition 3.1 (mot). *Soit A = {0, 1} l'alphabet. Le monoïde libre A^* est l'ensemble des mots finis de A. Un mot infini est un élément de A^N. L'ensemble des mots finis et infinis est noté par*

$$A^{\infty} = A^* \cup A^N.$$

Le mot vide est défini par ε *.*

Soit *w* un mot. La première lettre de *w* est notée *w*(*0*) et on note par *w*(*n-1*) (avec $w(n-1) \in A$) la $n^{\text{ième}}$ lettre de *w*.

Définition 3.2 (Décalage et rotation d'un mot). *On note par S le décalage (ou shift) d'un mot. C'est la fonction de A^{∞} vers A^{∞} telle que*

$$S(w(0)w(1)w(2)\ldots) = w(1)w(2)w(3)\ldots.$$

La composition du shift sera notée par S_m avec

$$S_m(w(1)w(2)w(3)\ldots) = w(m+1)w(m+2)w(m+3)\ldots$$

On note par $\Re$ *la rotation d'un mot. C'est la fonction de A^* vers A^* telle que*

$$\Re\,(w(0)w(1)w(2)\ldots w(n)) = w(n)w(0)w(1)\ldots w(n-1)$$

La composition de la rotation sera notée par

$$\Re_m(w(0)w(2)..w(n)\ ) = w(n-m+1)..w(1)..w(n-m)).$$

Définition 3.3 (Facteur d'un mot). *Soit un mot w (fini ou infini) de A. Le mot f est appelé un facteur du mot w s'il existe deux mots x et y (potentiellement vides) tels que* $w = x \cdot f \cdot y$ *.*

La pente d'un mot *w* fini, non vide, est définie par :

$$P(m) = \frac{|w|_1}{|w|},$$

où $|w|_1$ est le nombre de 1 de *w* et $|w|$ sa longueur.

3.2 Mots mécaniques

Un mot mécanique est plus souvent connu sous le nom de mot de Sturm. Dans ce qui suit, nous allons présenter la définition du mot mécanique et quelques de ses propriétés principales.

Définition 3.4 (Mots mécaniques). *Le mot mécanique supérieur de pente* α *et de décalage* θ (α *et* $\theta \in \mathbb{R}$ *), est le mot infini* $\overline{w}_{\alpha}^{\theta}$ *, dans lequel la* $n^{ième}$ *lettre est calculée par*

$$\overline{w}_{\alpha}^{\theta}(n) = \lceil (n+1)\cdot\alpha + \theta \rceil - \lceil n\cdot\alpha + \theta \rceil \quad \forall n \geq 0. \tag{3}$$

Le mot mécanique inférieur de pente α et de décalage θ est le mot infini $\underline{w}_\alpha^\theta$, dans lequel la $n^{ième}$ lettre : $\underline{w}_\alpha^\theta$ est calculée par

$$\underline{w}_\alpha^\theta(n) = \lfloor (n+1)\cdot\alpha + \theta \rfloor - \lfloor n\cdot\alpha + \theta \rfloor \quad \forall n \geq 0. \tag{4}$$

Lemme 3.5. *Soit $\overline{w}_\alpha^\theta$ le mot mécanique supérieur de pente α et de décalage θ. Si α est rationnel ($\alpha = p/q$), alors le mot est périodique et apériodique sinon.*

Par abus de langage, lorsqu'un mot est périodique nous ne considérons que sa plus petite période. Un mot mécanique supérieur fini (la plus petite période du mot de pente rationnelle) est une rotation du mot mécanique supérieur de décalage nul. C'est pourquoi, nous distinguons le mot mécanique supérieur de décalage nul, appelé mot mécanique supérieur et noté $\overline{w}_\alpha$, des mots mécaniques de décalage non nul appelés simplement mot mécaniques.

Lemme 3.6. *Le nombre de 1 de $\overline{w}_\alpha$ ($\alpha \in \mathbb{R}$) entre la lettre d'indice 0 et celle d'indice k égal*

$$\lceil (k+1)\cdot\alpha \rceil \quad k = 0,1,2,\ldots$$

Lemme 3.7. *Soit w_α le mot mécanique supérieur de pente α, ($\alpha \in R$), et de décalage nul. Pour tout $k \in \mathbb{N}$ et $k > 0$, l'indice de la $k^{ième}$ occurrence de la lettre '1' de w est égal à*

$$\left\lfloor \frac{k-1}{\alpha} \right\rfloor \tag{5}$$

Les preuves de ces résultats peuvent être trouvées dans [7].

Exemple 3.8. *Soit $\alpha = 3/5$, par les formules (3) et (4) on obtient*

$$\underline{w}_{3/5} = 01011, \quad \overline{w}_{3/5} = 11010.$$

Les nombres de lettre '1' de $\overline{w}_{3/5}$ jusqu'à la première lettre, la seconde lettre et la quatrième lettre valent respectivement $\lceil (0+1)\cdot(3/5) \rceil = 1$, $\lceil (1+1)\cdot(3/5) \rceil = 2$ et $\lceil (3+1)\cdot(3/5) \rceil = 3$. Les indices de la première occurrence et de la troisième occurrence de la lettre '1' de $\overline{w}_{3/5}$ sont respectivement $\lfloor 0\cdot 5/3 \rfloor = 0$ et $\lfloor 2\cdot 5/3 \rfloor = 3$.

Remarque 3.9. Soit $\overline{w}_\alpha$ un mot mécanique supérieur. La lettre d'indice *n* est en fait la $(n+1)^{ième}$ lettre du mot. Si la $k^{ième}$ occurrence de la lettre 1 a lieu à la $n^{ième}$ lettre, celle-ci aura pour indice *n*-1. Ceci provient du fait que les *n* premières lettres du mot sont indicées de 0 à *n*-1.

3.3 (*m*,*k*) patterns et mots mécaniques

Etudions les relations entre les (*m*,*k*) patterns et les mot mécaniques.

Théorème 3.10. *Le pattern de Ramanathan de paramètres (m,k) et le mot mécanique supérieur de pente m/k sont identiques.*

Démonstration : cf. Annexe.

Intéressons nous maintenant aux (*m,k*) patterns de Quan [8].

Corollaire 3.11. *Un (m,k) pattern de Quan est un mot mécanique de décalage non nul.*

Démonstration : On remarque que le mot calculé par la formule (2) est une rotation du pattern de Ramanathan. Comme le pattern de Ramanathan est le mot mécanique supérieur, le (*m,k*) pattern de Quan est une rotation du mot mécanique. □

Le fait de caractériser le pattern sous la forme d'un mot mécanique suggère que les politiques mécaniques font partie des suites de qualifications prédéfinies qui sont de bonnes approximations de politiques optimales. En effet, nous savons que les mots mécaniques répartissent les 1 d'une manière la plus régulière possible au sens où les écarts entre les 1 sont maximaux et ne prennent que deux valeurs $\lceil m/k \rceil$ $\lfloor m/k \rfloor$.

3.4 Nouvelle preuve de l'ordonnançabilité avec le (*m,k*) pattern de Ramanathan

En utilisant les propriétés des mots mécaniques, nous allons montrer qu'un certain nombre de résultats (lemmes et théorèmes) de Ramanathan [9] sont simplement des conséquences de propriétés des mots mécaniques.

Lemme 1 de [9]. *Pour chaque tâche τ_i, l'instance activée à l'instant nT_i, pour tout $n \in \mathbb{N}$, est marquée critique si et seulement s'il existe un nombre positif l tel que*

$$n = \left\lfloor l \cdot \frac{k_i}{m_i} \right\rfloor.$$

D'après le lemme 3.7, l'indice du $n^{\text{ième}}$ '1' dans $\overline{w}_{m_i/k_i}$ est donnée par $\lfloor (n-1)\cdot(k_i/m_i) \rfloor$. Ainsi le lemme 1 de [9] est une conséquence du lemme 3.7 en prenant $l = n$-1.

Lemme 2 de [9]. *Etant donné un ensemble de tâches $\tau_1, \tau_2 \ldots \tau_n$, l'algorithme de Ramanathan marque, pour chaque tâche τ_i, m_i instances critiques parmi les k_i premières instances.*

Lemme 3 de [9]. *Pour tout $1 \le i \le \mathbb{N}$, l'instance de τ_i activée à l'instant $(nT_i+k_iT_i)$ pour tout $n \in \mathbb{N}$ est marquée critique si et seulement si l'instance de τ_i activée à l'instant (nT_i) est aussi marquée critique.*

Par définition, un mot mécanique supérieur de pente m_i/k_i contient exactement m_i '1' et est de longueur k_i, ceci donne le lemme 2 de [9]. D'après le lemme 3.5, lorsque la pente d'un mot mécanique *m/k* est rationnelle, ce mot est périodique de période *k*, ce qui montre le lemme 3 de [9].

Grâce au lemme 3.5, nous pouvons généraliser le lemme 2 de [9].

Lemme 3.12. *Etant donné un ensemble de tâches $\tau_1, \tau_2 \ldots \tau_n$, la formule (1) marque, pour*

chaque tâche τ_i, m_i instances critiques parmi k_i instances consécutives quelconques.

Démonstration. On s'intéresse à tous les facteurs de longueur k_i du pattern infini. Comme le mot est périodique, ce nombre de facteurs est fini. De plus, tous ces facteurs sont des rotations de $\overline{w}_{m/k}$. Comme toutes les rotations sont des mots mécaniques de pente m_i/k_i, ceux-ci contiennent m_i lettres '1'. □

Lemme 4 de [9]. *Le nombre des instances critiques de la tâche τ_i dans l'intervalle $[nT_i, nT_i+b[$, $n \in \mathbb{N}$, $b \geq 0$, est maximum pour $n = 0$ et b quelconque.*

Démonstration. Soit b et $l \in \mathbb{N}$ tel que $b = lT_i$. Le nombre d'instances critiques entre deux lettres $w(n+l-1)$ et $w(n)$ est

$$\sum_{i=n}^{n+l-1} w(i) = \sum_{i=n}^{n+l-1} \lceil (i+1)\cdot\alpha \rceil - \lceil i\cdot\alpha \rceil = \lceil (n+l)\cdot\alpha \rceil - \lceil n\cdot\alpha \rceil$$

Comme pour tout $x, y \in \mathbb{R}$, $\lceil x \rceil + \lceil y \rceil \geq \lceil x+y \rceil$, on a $\lceil (n+l)\cdot\alpha \rceil - \lceil n\cdot\alpha \rceil \leq \lceil l\cdot\alpha \rceil = \sum_{i=0}^{l-1} w(i)$,

et le nombre d'instances critiques dans l'intervalle de temps $[nT_i, nT_i+b]$ est moins important que celui dans l'intervalle de temps $[0, b]$. Soit maintenant $l \in \mathbb{N}$ et $b \in \mathbb{R}$ tels que $(l-1)T_i \leq b < lT_i$. Comme, quelque soit n, le nombre d'instances critiques de la tâche est le même dans l'intervalle $[nT_i, nT_i+b[$ que dans l'intervalle $[nT_i, nT_i + lTi[$, on obtient le résultat. □

Théorème 3 de [9]. *Soit un ensemble de tâches $\tau_1, \tau_2 \ldots \tau_n$, tel que pour tout $T_1 < T_2 < \ldots < T_n$, on aie les termes suivants :*

$$R_{ij} = \left\{ \left\lfloor l \cdot \frac{k_j}{m_j} \right\rfloor T_j : \left\lfloor l \cdot \frac{k_j}{m_j} \right\rfloor T_j < T_i, l \in \mathbb{N} \right\}, \quad R_i = \bigcup_{j=1}^{i-1} R_{ij}$$

$$n_j(t) = \left\lceil \frac{m_j}{k_j} \left\lceil \frac{t}{T_j} \right\rceil \right\rceil, \tag{6}$$

$$W_i(t) = C_i + \sum_{j=1}^{i-1} n_j(t) \cdot C_j$$

Si pour tout $1 \leq i \leq n$, $t < T_i$, $\min_{t \in R_i} W_i(t)/t \leq 1$, alors la contrainte (m,k) peut être respectée pour toutes les tâches.

Démonstration : R_i donne les dates d'arrivée des instances critiques de toutes les tâches plus prioritaires que τ_i durant T_i. Le terme $n_j(t)$ représente le nombre d'instances critiques de la tâche τ_j arrivant dans l'intervalle $[0, t]$. Il est donné dans [9] sans justification car elle est considérée comme découlant d'un calcul complexe, alors que nous pouvons constater qu'elle découle simplement du lemme 3.6. m_j/k_j et $\lceil t/T_j \rceil$ représentent respectivement α et k+1 du lemme. $W_i(t)$ donne la quantité de travail total requise à l'instant t par l'instance critique de la tâche τ_i et les instances critiques de toutes les tâches plus prioritaires que τ_i. La condition $\min_{t \in R_i} W_i(t)/t \leq 1$ est une condition classique de faisabilité des systèmes temps réel. □

4. Ligne cellulaire et (*m*,*k*) pattern

Nous introduisons une nouvelles technique de partition des instances qui vise à distribuer différemment les 1. La construction des (*m*,*k*) patterns est maintenant basée sur des mots binaires appelés *lignes cellulaires.* Une ligne cellulaire peut être vue comme un mot fini périodique binaire apparaissant dans un certain nombre de problèmes d'optimisation [4] et qui se construit en temps linéaire. Les performances des (*m*,*k*) patterns peuvent être améliorées ensuite en utilisant des techniques heuristiques similaires à [8].

4.1 Ligne cellulaire

Soit $T_{m/k}$ l'ensemble des mots finis de longueur k qui comportent m '1'. Il existe au plus deux lignes cellulaires (une seule la plupart du temps) appartenant à $T_{m/k}$. Ces lignes cellulaires sont appelées lignes cellulaires de pente m/k et notées $C_{m/k}$. Dans ce qui suit et sans pertes de généralités, on s'intéresse à des ensembles tels que $m/k \leq 1/2$. Quand $m/k \leq 1/2$, alors la lettre '0' apparaît la plus fréquemment dans les mots. Lorsqu'on doit traiter des patterns tels que $2m > k$, on se ramène au cas $m/k \leq 1/2$ en échangeant les 0 et les 1. L'approche de [4] pour vérifier qu'un mot de $T_{m/k}$ est une ligne cellulaire permet de comprendre la manière dont la répartition des 1 s'effectue.

Définition 4.1 (Mot complètement partitionné). *Un mot fini w est dit complètement partitionné si les lettres '1' partitionnent w en*

- *(m+1) sous mots non vides composés uniquement de lettres '0' quand m/k* < 1/2 ;
- *m mots composés d'un '0' si m/k = 1/2.*

On définit Φ la transformation sur les mots complètement partitionnés, telle que $\Phi(w) = w^{(1)}$ avec $w^{(1)}$ obtenu par

1) Remplacement de chaque sous mot composés de 0 par sa longueur,

2) Suppression des symboles de minorité,

3) Remplacement des chiffres des longueurs par 0 et 1 : 0 pour la plus petite et 1 pour la plus grande. Lorsqu'il n'y a qu'une seule longueur, on remplace par 1.

4) Inversion des 0 et des 1 si nécessaire.

Le mot obtenu est le mot dérivé du premier ordre de w et il est noté $w^{(1)}$. Cette transformation peut être appliquée tant que le mot obtenu est complètement partitionné. Le mot obtenu après j transformations est appelé mot dérivé du $j^{\text{ième}}$ ordre. S'il existe un entier n, tel que $w^{(n)} = 1$, alors pour tout $i > n$, $w^{(i)}$ existe et $w^{(i)} = 1$.

Définition 4.2 (Ligne cellulaire [4]). *Un mot fini* $w \in T_{m/k}$ *est une ligne cellulaire, si pour tout* $j > 0$*, son mot dérivé du* $j^{ième}$ *ordre* $w^{(j)}$ *existe.*

Exemple 4.3. *On cherche la ligne cellulaire* $C_{2/6}$*. Considérons le mot complètement*

partitionné w = 010010. D'après la transformation illustrée ci-dessus, le mot 121 est obtenu en remplaçant chaque sous mot composé de 0 par sa longueur et en supprimant les symboles de minorité qui sont ici les lettres '1'. Puis les lettres '1 sont remplacées par '0' et la lettre '2' par '1', ainsi $w^{(1)} = 010$. *De la même façon, on obtient* $w^{(2)} = 11$, $w^{(j)} = 1$ pour tout $j \geq 3$. *Ainsi on vérifie que* $w = C_{6/2}$.

La construction d'une ligne cellulaire de pente m/k se fait à l'aide de l'algorithme présenté dans [2]. Cet algorithme détermine la ligne linéairement à la longueur du mot par une démarche inverse à la dérivation des mots : on détermine le premier mot dérivé réduit à 1 et on construit les mots dérivés d'ordre inférieur.

Ainsi étant donnée une contrainte (m,k)-firm nous construisons une ligne cellulaire de pente m/k. Cette ligne cellulaire est alors prise comme (m,k) pattern.

Une ligne cellulaire à l'instar des mots mécaniques répartit uniformément les lettres '1', mais à tendance à ne pas les placer aux extrémités, ainsi $w_{2/6}$=100100 et $C_{2/6}$=010010. Ceci aura comme conséquence de limiter les concentrations d'instances critiques aux mêmes instants, et permet d'améliorer l'ordonnançabilité et le temps de réponse moyen comme nous le verrons dans la section 5.

4.2 Amélioration de la performance du pattern de ligne cellulaire

L'interférence d'exécution joue un rôle important dans l'ordonnançabilité d'un ensemble de tâches, ce quelque soit la nature du (m,k) pattern. Cette interférence d'exécution peut être vue comme la perturbation générée par une tâche plus prioritaire. C'est pourquoi la méthode d'optimisation proposée dans [8] qui vise à réduire la plus grande interférence d'exécution peut aussi s'appliquer quand les (m,k) patterns sont des lignes cellulaires. C'est ce que nous allons faire ici.

Pour appliquer cet algorithme, une étape critique est de trouver les WCIP. Mais comme les lettres '1' sont distribuées différemment dans une ligne cellulaire, le WCIP n'existe pas toujours.

Exemple 4.4. *Le pattern de Ramanathan de pente 3/9 est 100100100. Le WCIP se trouve à la position du premier '1', car à partir de cette lettre, le nombre de lettres '1' est toujours le plus important dans tout intervalle par rapport tout autre intervalle de même longueur. Lorsque le pattern est le pattern de ligne cellulaire de pente 3/9 qui est 010010100, alors le nombre maximum de lettres '1' dans un intervalle de longueur 3 se trouve entre la* $4^{ième}$ *lettre et la* $6^{ième}$ *lettre, mais le nombre maximum de lettres '1' dans un intervalle de longueur 6 se trouve entre la deuxième lettre et la* $7^{ième}$ *lettre. Donc il n'existe pas de WCIP dans ce pattern.*

Ainsi pour pouvoir améliorer la performance du (m,k) pattern de ligne cellulaire, il faut trouver des instants critiques, à partir desquels, le nombre d'instances critiques devient important pour influer sur l'ordonnançabilité d'une tâche moins prioritaire. Nous définissons ainsi la notion de WCIP faible.

Définition 4.5 (WCIP faible). *Le WCIP faible d'une tâche* τ_i *est le début t d'un intervalle de temps de longueur l, qui satisfait les conditions suivantes :*

1) *le nombre d'instances critiques dans cet intervalle et tous ses sous-intervalle commençant à cet instant est le plus grand comparativement à tout autre intervalle de même longueur ;*
2) *il n'existe pas d'autre intervalle commençant à un instant différent et de longueur plus grande que l qui satisfait la condition 1.*

Exemple 4.6. *Le WCIP faible de la tâche τ_i , dont le pattern de ligne cellulaire $C_{3/9}$ est 010010100, se trouve à l'instant nT_i pour tout $n \in \{4, 4+9, 4+9+9...\}$. Car à partir de la $5^{ième}$ lettre du pattern, le nombre de lettres '1' est le plus important dans tout sous-intervalle de longueur plus petite que 4 comparativement à tout autre intervalle de même longueur. De plus il n'existe pas d'autre intervalle de longueur plus grande que 3 qui possède cette propriété.*

Lorsqu'un WCIP existe dans un (*m,k*) pattern, le WCIP faible correspond à ce WCIP.

Pour appliquer une démarche similaire à [8] on procède comme suit. On détermine la ligne cellulaire $C_{m/k}$ comme (*m,k*) pattern d'une tâche. On recherche ensuite les WCIP ou au cas échéant les WCIP faibles. Une fois les WCIP (ou les WCIPs faibles) déterminés, on utilise une approche similaire à [8] à l'aide des interférences d'exécution afin de calculer les valeurs de rotation des lignes cellulaires. A partir de ces valeurs, on obtient des rotations des lignes cellulaires qui sont alors prises comme patterns. Les WCIP (ou les WCIP faibles) entre deux tâches sont décalés le plus loin possible dans le temps.

Contrairement aux cas des patterns de Ramanathan, dans le cas de lignes cellulaires on ne peut pas garantir que la plus grande interférence d'exécution soit réduite après rotation. Ceci est dû à l'existence possible de plusieurs WCIP faibles. Deux WCIP faibles de tâches différentes peuvent alors être très proches rendant inefficace toute rotation. Ceci plaide pour de plus amples recherches sur ce point. Toutefois, les résultats expérimentaux vont montrer que cette approche nous permet d'obtenir de meilleures performances que celles de la littérature.

5. Résultats expérimentaux

Présentons des expérimentations qui vont permettre de comparer les performances de l'algorithme que nous avons implanté avec celles des méthodes proposées dans [8] et [9].

Nous considérons des ensembles de 3 tâches. La période de la tâche T_i est sélectionnée aléatoirement et distribuée uniformément entre 2 et 20. La valeur de k_i est distribuée aléatoirement et uniformément entre 2 et 10, et celle de m_i est distribuée de la même façon entre 1 et k_i-1. Le temps d'exécution C_i de chaque tâche est sélectionné aléatoirement entre 1 et T_i.

La charge totale du système est calculée en tenant compte des contraintes (*m,k*) par $\sum_{i=1}^{n} \frac{m_i C_i}{k_i T_i}$. Le temps de réponse moyen (temps passé par les instances dans le système) des instances critiques est la moyenne pondérée par m_i/k_i des moyennes de Césaro de tous les temps de réponse des instances critiques d'une tâche.

Nous traitons les simulations dans des intervalles de charge de longueur 0.2. Pour chaque

intervalle de charge, 5000 ensembles de tâches différents sont simulés.

Nous calculons aussi la performance des patterns dans lesquels les lettres ‘1’ sont distribuées aléatoirement pour la comparer avec les performances des patterns présentés. Pour ce faire, on génère pour chaque ensemble de tâches 500 (*m,k*) patterns aléatoires différents et la performance moyenne est calculée. Ceci permet d’obtenir une valeur moyenne pour l’ordonnançabilité et pour les temps de réponses.

	Nombre d’ensembles ordonnançables Temps de réponse moyen					Amélioration (%) de Ordonnançabilité Temps de réponse moyen	
Charge du système	Patterns aléatoires	Pattern de Ramanathan	Pattern de Quan	Pattern de Ligne cellulaire	Pattern de Ligne cellulaire Optimisé	Par rapport au pattern de Ramanathan	Par rapport au pattern de Quan
0.8 – 1.0	63 4.402	69 4.574	90 4.412	94.5 4.367	101 4.357	46.37 4.74	12.22 1.24
0.6 – 0.8	940.3 3.372	1095 3.506	1240 3.405	1213.5 3.435	1256 3.41	14.7 2.73	1.29 0.72
0.4 – 0.6	2839.5 2.454	3082 2.524	3215.5 2.466	3209.5 2.473	3242.5 2.464	5.2 2.37	0.83 0.08
0.2 – 0.4	4357.6 1.544	4413.5 1.604	4471 1.554	4468 1.554	4475 1.548	1.39 3.49	0.0008 0.38

Table 1. Les résultats expérimentaux

Dans ce tableau nous affichons pour chaque pattern et pour chaque intervalle de charge le nombre d’ensembles ordonnançables (valeur du dessus) et le temps de réponse moyen (valeur du dessous). Les deux dernières colonnes montrent la pourcentage d’amélioration des patterns de ligne cellulaire avec rotation par rapport aux patterns de Ramanthan et aux patterns de Quan pour le nombre d’ensembles ordonnançables (valeur du dessus) et pour le temps de réponse moyen (valeur du dessous). On constate que les lignes cellulaires optimisées ont de meilleures performances que les pattern de Quan et de Ramanathan que ce soit vis à vis de l’ordonnançabilité que du temps de réponse. Cette amélioration est d’autant plus forte que la charge est élevée. Pour les charges faibles l’amélioration est moins importante du fait de la plus grande marge de manœuvre dont on dispose pour l’ordonnançabilité. On constate également que le temps de réponse moyen est le plus faible pour des patterns aléatoires ce qui, pondéré par le nombre d’ensembles ordonnançables, laisse supposer que les politiques présentées préemptent plus souvent les tâches moins prioritaires (plus une instance est préemptée plus celle-ci reste longtemps dans le système).

6. Conclusion et perspectives

Dans ce travail nous avons d’abord montré que les (*m,k*) patterns introduits dans la littérature sont des mots mécaniques. La théorie des mots a permis de ré-exprimer différemment et plus simplement les preuves de résultats de [9]. Nous proposons ensuite une nouvelle technique de partition des instances dans un (*m,k*) patern à base de lignes cellulaires. Celle-ci, comme le montre les expérimentations, donne de meilleures performances. De plus,

comme l'approche présentée ici se base sur des suites prédéfinies facilement calculables, elle semble pouvoir être facilement implantées dans des serveurs et semble également pouvoir s'adapter en ligne aux variations de capacité du serveur (processeur d'un CPU, bande passante dans un réseau, …) en adaptant les paramètres *m* et *k* et en calculant les patterns rapidement. Nous pouvons ainsi conclure que la théorie des mots apporte effectivement une voie nouvelle pour l'analyse de l'ordonnançabilité sous contraintes (*m,k*)-firm avec (*m,k*) patterns fixes.

La relation entre pattern et théorie des mots exposée dans [7] pourrait être notablement améliorée en prenant en compte la relation qui unit mot mécanique et théorème des 3 longueurs [1]. Concernant l'ordonnancement des instances optionnelles, [3] qui utilise la politique Dual Priority basée sur des mots mécaniques semble également prometteur pour réduire le temps de réponse moyen des instances optionnelles tout en garantissant le respect des échéances des instances critiques. Enfin l'analyse de l'ordonnançabilité sous contraintes (*m,k*)-firm avec (*m,k*) patterns fixes mais avec échéances quelconques, ainsi que l'ordonnancement sans préemption font partie de nos travaux futurs.

8. Bibliographie

[1] P. Alessandri and V. Berthé. "Three distance theorems and combinatorics on words", *L'enseignement mathématique*, (44) pp103-132, 1998.

[2] B. Gaujal and E. Hyon "Factorization of mechanical words and continued fractions", *WACAM 2004*, pp. 37-43, 2004.

[3] B. Gaujal and N. Navet "Trafic shaping in real time distributed systems : a low complexity approach" *Computer Communication*, 1999.

[4] P. Geer, H. W. McLaughlin, and K. Unsworth "Cellular line : An Introduction" *2nd Conference on Discrete Mathematics and Computer Science*, pp167-178, 2003.

[5] M. Hamdaoui and P. Ramanathan "A dynamic priority assignment technique for streams with (*m,k*)-firm deadlines" *IEEE Transactions on Computers*, 44:1443-1451, Dec 1995.

[6] C.L. Liu and J.W. Layland "Scheduling algorithms for multiprogramming in hard real-time environment" *Journal of the ACM*, 20(1) : pp40-61, 1973.

[7] M. Lothaire "Algebraic Combinatorics on Words", *Cambridge University Press*, 2002.

[8] G. Quan and X. Hu "Enhanced Fixed-priority Scheduling with (*m,k*)-firm Guarantee" *Proc. Of 21st IEEE Real-Time Systems Symposium*, pp.79-88, 2000.

[9] P. Ramanathan "Overload management in Real-Time control applications using (*m,k*)-firm guarantee" *IEEE Transactions on Parallel and Distributed Systems*, 10(6) :549-559, Jun 1999.

[10] Y.Q Song et A. Koubâa, "Gestion dynamique de la Qds temps réel selon (*m,k*)-firm" *Ecole d'été temps réel, ETR 2003*, pp327-342, Toulouse, 9-12 sept. 2003.

[11] A. Koubâa and Y.Q. Song, "Loss-Tolerant QoS using Firm Constraints in Guaranteed

Rate Networks", *10th IEEE Real-Time and Embedded Technology and Applications (RTAS'2004)*, pp526-533, Toronto (Canada) 25-28 May 2004.

Annexe: Démonstration du théorème 3.10.

Montrons tout d'abord que le (*m*,*k*) pattern de Ramanathan est un mot mécanique supérieur de pente *m*/*k*.

Supposons que la lettre d'indice *n* du (*m*,*k*) pattern de Ramanathan est '1'. Par la formule (1), la lettre d'indice *n* est '1' si pour $0 < \alpha < 1$, on a

$$n = \lfloor \lceil n \cdot \alpha \rceil \cdot (1/\alpha) \rfloor .$$

Il vient alors

$$n \le \lceil n \cdot \alpha \rceil \cdot (1/\alpha) < n+1.$$

Cela donne $n \cdot \alpha \le \lceil n \cdot \alpha \rceil < (n+1) \cdot \alpha.$ On obtient donc

$$\lceil n \cdot \alpha \rceil < \lceil (n+1) \cdot \alpha \rceil .$$

Comme pour tout $0 < \alpha \le 1$ et pour tout nombre entier *n*, on a $\lceil (n+1) \cdot \alpha \rceil - \lceil n \cdot \alpha \rceil \in \{0, 1\}$. Il vient ainsi

$$\lceil (n+1) \cdot \alpha \rceil - \lceil n \cdot \alpha \rceil = 1.$$

Donc si la lettre d'indice *n* du (*m*,*k*) pattern de Ramanathan est '1', la lettre d'indice *n* de $\overline{w}_{m/k}$ l'est aussi.

Supposons maintenant que la lettre d'indice *n* du (*m*,*k*) pattern de Ramanathan est '0'. Si la lettre d'indice *n* est '0' dans un mot obtenu par la formule (1), la condition $n \ne \lfloor \lceil n \cdot \alpha \rceil \cdot (1/\alpha) \rfloor$ est alors vérifiée. Comme *n* est un entier, alors une des deux inégalités suivantes doit être vérifiée

$$\lceil n \cdot \alpha \rceil \cdot (1/\alpha) \ge n+1 \qquad \text{ou} \qquad \lceil n \cdot \alpha \rceil \cdot (1/\alpha) < n .$$

La condition $\lceil n \cdot \alpha \rceil \cdot (1/\alpha) < n$ est irréalisable, car $\lceil n \cdot \alpha \rceil \ge n \cdot \alpha$. Donc, c'est l'inégalité $\lceil n \cdot \alpha \rceil \cdot (1/\alpha) \ge n+1,$ qui est vérifiée, celle-ci donne

$$\lceil n \cdot \alpha \rceil \ge (n+1) \cdot \alpha . \tag{7}$$

D'autre part, comme $\lceil (n+1) \cdot \alpha \rceil - (n+1) \cdot \alpha < 1$, on obtient

$$\lceil (n+1) \cdot \alpha \rceil - 1 < (n+1) \cdot \alpha. \tag{8}$$

Ainsi de (7) et (8), il vient $\lceil n \cdot \alpha \rceil > \lceil (n+1) \cdot \alpha \rceil - 1,$ ce qui donne

$$\lceil (n+1) \cdot \alpha \rceil - \lceil n \cdot \alpha \rceil < 1.$$

La seule valeur possible pour cette inégalité est 0. Donc la lettre d'indice *n* de $\overline{w}_{\alpha}$ est 0.

Ainsi le (*m*,*k*) pattern de Ramanathan est le mot mécanique supérieur de pente *m*/*k*.

Montrons maintenant que le mot mécanique supérieur de pente *m*/*k* est le pattern de Ramanathan pour la contrainte (*m,k*).

On suppose que la lettre considérée est la $k^{ième}$ occurrence de la lettre '1' de $\overline{w}_\alpha$. Selon le lemme 3.7, on a

$$n = \lfloor (k-1)/\alpha \rfloor.$$

D'après le lemme 3.6, le nombre de lettres '1' jusqu'à la lettre d'indice *n*-1 est $\lceil n \cdot \alpha \rceil$, on obtient alors

$$n = \lfloor \lceil n \cdot \alpha \rceil \cdot (1/\alpha) \rfloor.$$

Donc, la lettre d'indice *n* de pattern de Ramanathan est aussi '1'.

Supposons que la lettre d'indice *n* de $\overline{w}_\alpha$ est '0'. Si $\overline{w}_\alpha(n) = 0$ on a

$$\lceil (n+1) \cdot \alpha \rceil = \lceil n \cdot \alpha \rceil.$$

D'après le lemme 3.6, le nombre de '1' de $\overline{w}_\alpha$ jusqu'à la lettre d'indice *n* vaut $\lceil (n+1) \cdot \alpha \rceil$. D'autre part, d'après le lemme 3.7, $\lfloor \lceil n \cdot \alpha \rceil / \alpha \rfloor$donne l'indice du $(\lceil n \cdot \alpha \rceil + 1)^{ième}$ '1' du mot. Comme $\lceil (n+1) \cdot \alpha \rceil = \lceil n \cdot \alpha \rceil$ et que le nombre de '1' jusqu'à la lettre d'indice *n* est $\lceil (n+1) \cdot \alpha \rceil$, on sait alors que l'indice de la $(\lceil n \cdot \alpha \rceil + 1)^{ième}$ '1' se trouve après l'indice *n*. Ainsi $\lfloor \lceil n \cdot \alpha \rceil / \alpha \rfloor > n$ et la lettre d'indice *n* du pattern de Ramanathan est '0' d'après la formule (1).

Ainsi un mot mécanique de pente *m*/*k* est un (*m*,*k*) pattern de Ramanathan.

Donc le (*m,k*) pattern de Ramanathan et le mot mécanique supérieur sont identiques. □