ANNEE 2021

SCIENCES SORBONNE UNIVERSITÉ

RAPPORT DE STAGE

présenté par

BIEGAS Emilie

Mission effectuée du 28/06/2021 au 31/08/2021 au :

LIP6 de Sorbonne Université

Sujet de la mission :

Algorithmes pour des problèmes d'optimisation combinatoire

Tuteur de stage : **BAMPIS Evripidis**

Lien d'accès au GitHub :
https ://github.com/EmilieBiegas/Learning-Augmented-Algorithms

Table des matières

1 Introduction

1.1 Cadre du stage et résumé

Ce stage a été effectué au LIP6 de l'université Sorbonne Sciences pour une durée de 46 jours à temps complet (entre le 28 juin et le 31 août 2021) dans l'encadrement de Mr Evripidis Bampis, professeur-chercheur de l'équipe RO du LIP6 de Sorbonne Université. Il a été effectué en présentiel pendant les 3 premières semaines avec un rendez-vous par semaine avec le tuteur (dans son bureau en présentiel) puis à distance jusqu'à la fin du stage, avec un accès à des bureaux au LIP6. Ce stage m'a également permis de communiquer avec toute l'équipe RO du LIP6 de Sorbonne Université et ainsi m'informer sur le travail des autres stagiaires. Le sujet d'origine du stage était "Algorithmes pour des problèmes d'optimisation combinatoire" et concernait plus précisément les Learning Augmented Algorithms présentés ci-dessous. Le cahier des charges était donc constitué de l'étude bibliographique et de la compréhension du sujet (prévue pour les 2 premières semaines), de la conception de solutions algorithmiques (prévue pour les 5 semaines centrales) et enfin de l'évaluation analytique ou par simulation (prévue pour les 2 dernières semaines). Nous avons effectivement lors de ce stage commencé par faire une étude bibliographique puis conçu des solutions algorithmiques et finalement effectué des évaluations analytiques ainsi que des simulations.
En effet, après avoir étudié ce qui a été fait sur l'exemple du problème de la rentabilité des skis, nous avons essayé d'adapter ces méthodes au problème d'économie d'énergie pour une machine à états. Plus précisément, les approches simple hors-ligne et en ligne, aveugle avec prédictions, robuste et consistant avec prédictions, et Primal-Dual avec et sans prédictions pour le cas en ligne ont été détaillées pour le problème de la rentabilité des skis. Lors de ce stage, nous avons tout d'abord complété les informations concernant le problème de la rentabilité des skis en reformulant la preuve complète du coût de l'algorithme Primal-Dual avec prédictions, en encodant les algorithmes précédemment cités et en effectuant des simulations afin de comparer les différentes approches. Ce stage a ensuite permis de faire le lien entre ce problème et le problème d'économie d'énergie pour une machine à 2 états puis d'étendre le problème à 3 puis à k états et ainsi d'adapter les méthodes précédemment citées à ces problèmes. Nous avons ainsi mis en forme les programmes linéaires primal et dual pour le problème d'économie d'énergie pour une machine à états, puis adapté les algorithmes simple hors-ligne et en ligne, aveugle avec prédictions, et robuste et consistant avec prédictions à ces problèmes. Nous avons ensuite encodé ces algorithmes pour le cas à 3 et k états puis mis en place des simulations afin de comparer les algorithmes pour chacun des deux problèmes. Nous avons également formulé le ratio de compétitivité de l'algorithme robuste et consistant avec prédiction pour le problème d'économie d'énergie pour une machine à 2 états et mis en forme la preuve. Enfin, nous avons réfléchi à la preuve de robustesse pour l'algorithme simple du problème d'économie d'énergie pour une machine à 3 états. Puis, nous avons réfléchi à la formulation des approches Primal-Dual pour le problème d'économie d'énergie pour une machine à 3 et k états, encodé les algorithmes associés et effectué des simulations afin de comparer cette approche avec les précédentes.
NB1 : Vous pouvez retrouver les fichiers code en suivant le lien GitHub indiqué sur la page de garde.
NB2 : Les informations des paragraphes 1.2, 1.3, et 2.1 sont des états de l'art et proviennent intégralement (après traduction) des articles [1], [2], et [3]. De même, les informations présentées dans le paragraphe 3.1 sont tirés des articles [5] et [8].

1.2 Introduction du sujet

L'amélioration des performances d'un algorithme en ligne (*online*) grâce à des prédictions présente un domaine actif de recherche. Le principe est de combiner un algorithme performant connaissant les vraies valeurs de ce qui est prédit et un algorithme qui reste bon même dans le pire cas. Ainsi, si les prédictions sont exactes, l'algorithme aura des performances proches des meilleurs performances pour un algorithme hors ligne (*offline*) de ce problème (consistance grâce au premier algorithme combiné) et aura des performances proches de celles d'un algorithme en ligne sans prédiction lorsque les prédictions sont erronées (robustesse grâce au second algorithme combiné). On appelle ces algorithmes les *Learning Augmented Algorithms*.
Nous allons donc étendre la méthode Primal-Dual pour les algorithmes en ligne au cas où des prédictions sont introduites.

1.3 Définitions

Un algorithme en ligne (*online*) est, à l'inverse d'un algorithme hors ligne (*offline*), un algorithme dont l'entrée est visible partie par partie et non pas d'un coup. Un algorithme en ligne va donc devoir prendre une décision à chaque nouvelle information reçue en entrée et ne pourra jamais revenir sur une décision passée. Par exemple, lorsque la version hors ligne de l'algorithme traitant du problème de la rentabilité des skis (*The Ski rental problem*) prendra en paramètre le nombre de jours de vacances, l'algorithme en ligne traitant du même problème traitera les jours un par un sans savoir combien de jours il reste à venir et sans pouvoir revenir sur

une décision prise à un jour précédent.
Dans ce rapport, nous noterons η l'erreur de prédiction totale et η_i l'erreur de prédiction sur la valeur i, c'est à dire $|x_i - y_i|$, où x_i est la vraie valeur et y_i la valeur prédite.
Le ratio de compétitivité d'un algorithme est défini comme le coût de l'algorithme dans le pire cas divisé par le coût optimal dans la version hors ligne du problème. Ce dernier est une fonction c(η) des erreurs η de prédiction. Un algorithme est γ-robuste si c(η) $\leq \gamma$ pour tout η, c'est à dire que la robustesse mesure les performances dans le pire cas, avec les pires prédictions. Un algorithme est β-consistant si c(0) = β, c'est à dire que la consistance mesure les performances dans le cas où les prédictions sont exactes.
Un algorithme ALG est c-compétitif si pour n'importe quelle entrée I, le coût $c_{ALG}(I)$ de l'algorithme ALG sur l'entrée I satisfait $c_{ALG}(I) \leq c \,.\, OPT(I)$, où $OPT(I)$ est le coût optimal de l'algorithme hors ligne.

2 Le problème de la rentabilité des skis

2.1 Définition du problème et algorithmes de résolution

Le problème de la rentabilité des skis (*The Ski rental problem*) est le suivant : une personne est en vacances en montagne et se demande si elle doit louer des skis lors de son séjour (au prix de 1€ par jour de location) ou bien les acheter (au prix de B€).
Dans la version hors ligne de ce problème, on connaît le nombre de jours de vacances N. On peut ainsi établir un algorithme optimal déterministe qui décidera de louer les skis si N<B (pour un coût de N€) et de les acheter sinon (pour un coût de B€).
Au contraire, dans la version en ligne de ce problème, on ne connaît pas à l'avance le nombre de jours de vacances. On doit donc prendre une décision à chaque nouveau jour qui arrive sans remettre en cause les précédentes décisions. On peut ainsi établir un algorithme optimal déterministe qui décidera de louer les skis pendant les B premiers jours et de les acheter si on atteint la date B+1. En effet, ces choix seront optimaux si finalement le nombre de jours n'excède pas B et sera au pire le double de l'optimal si ce n'est pas le cas puisque dans l'idéal on aurait du acheter les skis dès le premier jour pour un coût de B€ et que finalement on a payé 2B€ (B jours de location plus l'achat des skis au jour d'après). Cet algorithme est donc 2-approché.
On peut d'ailleurs retrouver ce résultat grâce à la méthode Primal-Dual. Pour ce faire, on commence par mettre le problème sous forme de programme linéaire en nombre entier en posant $x = \begin{cases} 1 & \text{si on décide d'acheter les skis} \\ 0 & \text{sinon} \end{cases}$ et $z_j = \begin{cases} 1 & \text{si on décide de louer les skis au jour j} \\ 0 & \text{sinon} \end{cases}$. On a ainsi le programme linéaire suivant ainsi que son dual :

$$Primal : \begin{cases} Min\ B.x + \sum_{j=1}^{N} z_j \\ \text{Sous contraintes :} \\ \text{Pour chaque jour j, } x + z_j \geq 1 \\ x \geq 0, z_j \geq 0 \text{ pour chaque jour j} \end{cases} \quad (1) \qquad Dual : \begin{cases} Max\ \sum_{j=1}^{N} y_j \\ \text{Sous contraintes :} \\ \sum_{j=1}^{N} y_j \leq B \\ 0 \leq y_j \leq 1 \text{ pour chaque jour j} \end{cases} \quad (2)$$

On note qu'étant donné que l'on ne peut pas remettre en cause les décisions antérieures, les variables sont non décroissantes de manière monotone (on peut faire passer une variable de 0 à 1 mais pas l'inverse puisque si on a choisit d'acheter les skis ou de les louer, on ne peut pas faire marche arrière). On a donc à chaque nouveau jour une nouvelle variable duale et une nouvelle contrainte dans le primal à traiter. Le reste de la preuve se trouve à la page 111 du document [1]. Notons que l'on peut aussi randomiser le procédé afin d'améliorer les résultats.
On introduit pour les algorithmes suivants la fonction e(z) = $(1 + \frac{1}{B})^{z.b}$ qui tend vers e^z lorsque B tend vers l'infini.

Algorithme Primal-Dual sans prédictions

Initialisation : $x \leftarrow 0$, $z_j \leftarrow 0 \ \forall$ j
c $\leftarrow$ e(1), c' $\leftarrow 1$
Pour chaque nouveau jour j tel que $x + z_j < 1$:
    /*Mise à jour du primal
    $z_j \leftarrow 1 - x$
    $x \leftarrow (1 + \frac{1}{B})x + (\frac{1}{(c-1).B})$
    /*Mise à jour du dual
    $y_j \leftarrow$ c'

Plaçons nous désormais dans le cas où l'on possède une approximation du nombre de jours de vacances N_{pred} (toujours dans la version en ligne du problème évidemment). De la même manière que dans le cas précédent, à chaque nouveau jour j, une nouvelle contrainte $x + z_j \geq 1$ apparaît dans le primal. Pour satisfaire cette contrainte, l'algorithme met à jour les variables duales et primales afin d'obtenir une solution réalisable pour ces deux problèmes et en tentant de maintenir le ratio $\frac{\Delta P}{\Delta D}$ aussi petit que possible, où ΔP est l'augmentation du coût du primal suite aux mises à jour des variables et ΔD celle du dual. Pour ce faire, l'algorithme va se servir de la prédiction N_{pred} afin d'ajuster le taux de variation des variables. En particulier, lorsque $N_{pred} \geq B$, l'algorithme va augmenter la variable plus fortement que le fait l'algorithme traitant du problème en ligne sans prédiction. Cette augmentation va s'effectuer de manière plus ou moins drastique en fonction du paramètre de robustesse λ. Intuitivement, λ indique notre confiance en la prédiction, plus λ sera petit plus on aura confiance en la prédiction. L'algorithme est le suivant :

Algorithme Primal-Dual avec prédictions

Entrée : λ, N_{pred}
Initialisation : $x \leftarrow 0$, $z_j \leftarrow 0 \ \forall$ j
Si $N_{pred} \geq B$, alors :
    /*Les prédictions suggèrent d'acheter
    c $\leftarrow$ e(λ), c' $\leftarrow 1$
Sinon :
    /*Les prédictions suggèrent de louer
    c $\leftarrow$ e($\frac{1}{\lambda}$), c' $\leftarrow \lambda$

Pour chaque nouveau jour j tel que $x + z_j < 1$:
    /*Mise à jour du primal
    $z_j \leftarrow 1 - x$
    $x \leftarrow (1 + \frac{1}{B})x + (\frac{1}{(c-1).B})$
    /*Mise à jour du dual
    $y_j \leftarrow$ c'

On note que puisque $\lambda \leq 1$, alors $\lambda^2 \leq 1$, d'où $\lambda \leq \frac{1}{\lambda}$ et donc e(λ) $\leq$ e($\frac{1}{\lambda}$), d'où une plus grande augmentation de x si $N_{pred} \geq B$.

Théorème : pour n'importe quel $\lambda \in [0, 1]$, le coût de l'algorithme Primal-Dual avec prédictions, noté $c_{PDAP}(N_{pred}, I, \lambda)$, est borné comme suit :

$$c_{PDAP}(N_{pred}, I, \lambda) \leq min\{\frac{\lambda}{1 - e(-\lambda)}.S(N_{pred}, I), \frac{1}{1 - e(-\lambda)}.OPT(I)\}$$

où $S(N_{pred}, I)$ est le coût obtenu en suivant aveuglement les prédictions et $OPT(I)$ est le coût optimum. Dans notre cas on a donc $S(N_{pred}, I) = \begin{cases} B & \text{si } N_{pred} \geq B \\ N & \text{sinon} \end{cases}$.
Démonstration : La démonstration de ce théorème se trouve en annexe dans la section 6.1.
Une méthode plus simple serait, toujours dans le cas où l'on possède une approximation du nombre de jours de vacances N_{pred}, de suivre aveuglement les prédictions. On aurait alors l'algorithme suivant :

Algorithme aveugle avec prédictions

Entrée : N_{pred}
Si $N_{pred} \geq B$, alors :
    Acheter dès le premier jour pour un coût de B
Sinon :
    Louer tous les jours pour un coût de N

Malgré sa simplicité, cet algorithme est consistant puisque lorsque les prédictions sont exactes, il retourne la solution optimale. Cependant, cet algorithme n'est pas robuste. En effet, si la prédiction est très mauvaise, le coût obtenu sera très loin de l'optimal (le ratio de compétitivité sera très mauvais).
On pourrait donc imaginer, à partir de cet algorithme simple, un algorithme qui serait à la fois consistant et robuste en introduisant un hyperparamètre λ (qui est, comme précédemment, le paramètre de robustesse). On aurait alors l'algorithme suivant :

Algorithme robuste et consistant avec prédictions

Entrée : λ, N_{pred}
Si $N_{pred} \geq B$, alors :
    Acheter au début du jour $\lceil \lambda B \rceil$
Sinon :
    Acheter au début du jour $\lceil \frac{B}{\lambda} \rceil$

Cet algorithme a un ratio de compétitivité d'au plus $min\{\frac{1+\lambda}{\lambda}, 1+\lambda+\frac{\eta}{(1-\lambda).OPT(I)}\}$, où $OPT(I)$ est le coût optimum.
Démonstration : La preuve de l'obtention de ce ratio se trouve en annexe dans la section 6.2.
Pour conclure, cet algorithme donne un moyen simple et efficace de faire un compromis entre robustesse et consistance. En particulier, une grande confiance envers la prédiction se traduira par un λ proche de zéro, ce qui amène un meilleur ratio de compétitivité lorsque η est petit. D'un autre côté, lorsque λ est proche de un, cela caractérise une certaine méfiance envers les prédictions et assure la robustesse de l'algorithme. Notons qu'il est également possible de randomiser cet algorithme afin d'améliorer les performances.

2.2 Simulations

Comparons désormais les performances des algorithmes simples avec celles de l'approche Primal-Dual. Pour ce faire, nous avons effectué des simulations sur 15 000 exemples. Nous avons échantillonné l'axe des X en 100 parties égales. Pour le choix des données, nous avons pris un B égal à 15 et un nombre de jour maximum à 30. Nous tirons donc uniformément N entre 1 et 30 puis pour la prédiction nous prenons le N précédemment obtenu auquel nous ajoutons un entier aléatoire entre 0 et 30-N et nous retirons un nombre aléatoire entre 0 et N-1. Ainsi nous obtenons un Npred entre 1 et 30 qui est probablement proche du N.

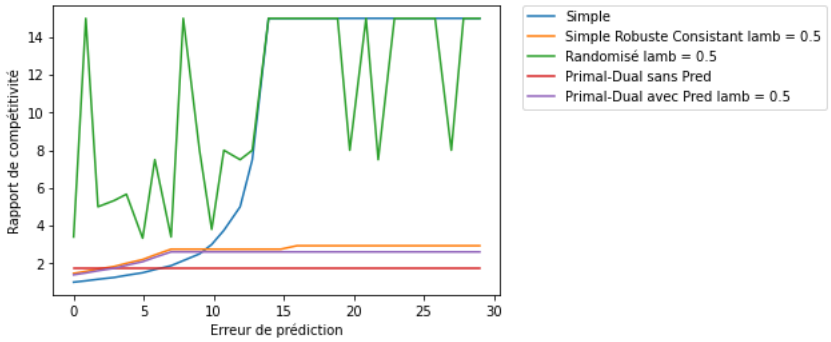

FIGURE 1 – Comparaison des algorithmes en fonction de l'erreur de prédiction

Nous voyons donc bien d'intérêt de l'approche Primal-Dual par rapport aux approches simples. En effet, lorsque les prédictions sont imparfaites (erreur de prédiction supérieure à 0), l'approche simple devient très vite

mauvaise (avec un rapport de compétitivité dépassant vite 10) alors que l'approche Primal-Dual fait preuve d'une certaine stabilité. En effet, l'algorithme Primal-Dual sans prédiction semble avoir un rapport de compétitivité de 2. L'algorithme Primal-Dual avec prédiction et l'algorithme simple robuste et consistant semblent avoir plus ou moins le même rapport de compétitivité pour un même lambda, bien que l'approche Primal-Dual semble être légèrement meilleure ici encore. L'algorithme randomisé à, quant à lui, un rapport de compétitivité assez aléatoire.

3 Problème d'économie d'énergie pour une machine à états

3.1 Pour une machine à 2 états

Soit une machine présentant deux états : ON et OFF. Rester dans l'état ON coûte A à chaque unité de temps et rester en OFF ne coûte rien. Par contre, lorsqu'une tâche arrive, il ne coûte rien de l'exécuter si on est dans l'état ON et il coûte C de l'exécuter si on était dans l'état OFF (coût du passage de OFF à ON). On note que l'on ne peut pas passer de l'état OFF à l'état ON avant l'arrivée de la tâche puisque cela aurait pour conséquence d'augmenter le coût (on aurait, pour chaque unité de temps, A à payer en plus puisque la machine serait dans l'état ON une unité de temps supplémentaire) et ce n'est pas souhaitable. Le problème est donc d'économiser l'énergie entre deux arrivées de tâches.
Ce problème serait simple si on connaissait les dates d'arrivées des tâches. Plaçons nous dans le cas où une tâche vient de finir et la prochaine tâche arrive dans N unités de temps. Un algorithme déterministe optimal serait alors le suivant : si on devait attendre moins de $\frac{C}{A}$ unités de temps, on laisserait la machine dans l'état ON pour un coût de $N.A \leq \frac{C}{A}.A = C$ et sinon on passerait directement à l'état OFF jusqu'à l'arrivée de la prochaine tâche pour un coût de C.
Dans le cas en ligne où l'on n'aurait aucune information concernant la durée séparant les deux tâches, un algorithme déterministe consisterait à rester dans l'état ON jusqu'au temps $\frac{C}{A}$ puis de passer à l'état OFF jusqu'à l'arrivée de la prochaine tâche.
Dans le cas où l'on possède une approximation du nombre d'unités de temps entre deux tâches N_{pred}, on pourrait, de la même manière qu'avec le problème de la rentabilité des skis, suivre aveuglement les prédictions. On aurait alors l'algorithme suivant :

Algorithme aveugle avec prédictions

Entrée : N_{pred}
Si $N_{pred} \geq \frac{C}{A}$, alors :
    Passer dans l'état OFF dès le premier instant pour un coût de C
Sinon :
    Rester dans l'état ON jusqu'à l'arrivée d'une nouvelle tâche pour un coût de $A.N$

Malgré sa simplicité, cet algorithme est consistant puisque lorsque les prédictions sont exactes, il retourne la solution optimale. Cependant, cet algorithme n'est pas robuste. En effet, si la prédiction est très mauvaise, le coût obtenu sera très loin de l'optimal (le ratio de compétitivité sera très mauvais).
On pourrait donc imaginer, à partir de cet algorithme simple, un algorithme qui serait à la fois consistant et robuste en introduisant un hyperparamètre λ (qui est, comme précédemment, le paramètre de robustesse). On aurait alors l'algorithme suivant :

Algorithme robuste et consistant avec prédictions

Entrée : λ, N_{pred}
Si $N_{pred} \geq \frac{C}{A}$, alors :
    Passer dans l'état OFF à l'instant $\lceil \lambda \frac{C}{A} \rceil$
Sinon :
    Passer dans l'état OFF à l'instant $\lceil \frac{C}{\lambda A} \rceil$

Cette formulation traduit bien une confiance totale en la prédiction si λ vaut zéro et une méfiance grandissante lorsqu'il se rapproche de un. En effet, on note tout d'abord que $\lceil \lambda \frac{C}{A} \rceil \leq \frac{C}{A}$ et $\lceil \frac{C}{\lambda A} \rceil \geq \frac{C}{A}$, ce qui traduit bien le fait que lorsqu'il est prédit que le nombre d'unité de temps n'excède pas $\frac{C}{A}$, on va retarder le passage à l'état OFF contrairement au cas où l'on prédit un nombre d'unité de temps plus important. On remarque ensuite

que, lorsque λ est nul, c'est à dire lorsque les prédictions sont supposées excellentes, l'algorithme ci-dessus est équivalent à l'algorithme aveugle : si $N_{pred} \geq \frac{C}{A}$, alors passer dans l'état OFF à l'instant 0 ; et sinon, ne jamais passer dans l'état OFF. De même, lorsque λ vaut 1, c'est à dire lorsque les prédictions sont supposées totalement incorrectes, l'algorithme reste plutôt bon et reprends certaines caractéristiques de l'algorithme qui consistait simplement à passer à l'état OFF au pas de temps $\frac{C}{A}$ afin d'avoir un algorithme 2-approché.
En effet, dans la version en ligne de ce problème, on ne connaît pas à l'avance le nombre d'unités de temps séparant deux tâches. On doit donc prendre une décision à chaque nouveau pas de temps qui arrive sans remettre en cause les précédentes décisions. On peut ainsi établir un algorithme optimal déterministe qui décidera de rester dans l'état ON pendant les $\frac{C}{A}$ premiers instants (pour un coût de C) et de passer dans l'état OFF si on atteint le pas de temps $\frac{C}{A} + 1$. En effet, ces choix seront optimaux si finalement le nombre de jours n'excède pas $\frac{C}{A}$ et sera au pire le double de l'optimal si ce n'est pas le cas puisque dans l'idéal on aurait du passer dans l'état OFF dès le premier jour pour un coût de C et que finalement on a payé $2C$ (on paye A pendant les $\frac{C}{A}$ premiers instants donc un coût de C puis le coût de C pour passer de l'état OFF vers l'état ON lors de l'arrivée de la nouvelle tâche). Cet algorithme est donc 2-approché.
Cet algorithme a un ratio de compétitivité d'au plus $min\{\frac{1+\lambda}{\lambda}, 1 + \lambda + \frac{\eta}{(1-\lambda).OPT(I)}\}$, où $OPT(I)$ est le coût optimum.
Démonstration : La preuve de l'obtention de ce ratio se trouve en annexe dans la section 6.3.
Pour conclure, cet algorithme donne un moyen simple et efficace de faire un compromis entre robustesse et consistance. En particulier, une grande confiance envers la prédiction se traduira par un λ proche de zéro, ce qui amène un meilleur ratio de compétitivité lorsque η est petit. D'un autre côté, lorsque λ est proche de un, cela caractérise une certaine méfiance envers les prédictions et assure la robustesse de l'algorithme. Notons qu'il est également possible de randomiser cet algorithme afin d'améliorer les performances.
Maintenant que nous avons formulé les deux algorithmes simples dans le cadre de ce problème, intéressons nous à l'approche Primal-Dual. Nous cherchons donc à résoudre le programme linéaire suivant :

$$Primal : \begin{cases} Min\ C.x + \sum_{i=1}^{N} A.t_i \\ \text{Sous contraintes :} \\ \text{Pour chaque unité de temps i, } x + t_i \geq 1 \\ x \geq 0, t_i \geq 0 \text{ pour chaque unité de temps i} \end{cases} \quad (3)$$

$$Dual : \begin{cases} Max\ \sum_{i=1}^{N} y_i \\ \text{Sous contraintes :} \\ \sum_{i=1}^{N} y_i \leq C \\ \sum_{i=1}^{N} y_i \leq A \\ 0 \leq y_i \leq 1 \text{ pour chaque unité de temps i} \end{cases} \quad (4)$$

où $x = \begin{cases} 1 & \text{si on décide de passer dans l'état OFF} \\ 0 & \text{sinon} \end{cases}$, $t_i = \begin{cases} 1 & \text{si la machine est dans l'état ON au temps i} \\ 0 & \text{sinon} \end{cases}$,
et N est le nombre d'unité de temps qui s'écoule entre la fin de la dernière tâche et l'arrivée d'une nouvelle tâche.
Dans la version en ligne de ce problème, c'est-à-dire lorsque l'on ne connaît pas la valeur de N, à chaque nouveau pas de temps, une variable t_i et sa contrainte d'intégrité associée s'ajoutent au problème et on doit donc prendre une décision (laisser la machine dans l'état dans lequel elle est ou, si elle était dans l'état ON, choisir de faire une transition vers l'état OFF) à chaque pas de temps sans remettre en cause les décisions précédentes.
La contrainte d'intégrité permet à la fois d'exprimer le fait que la machine ne peut pas être ni dans l'état ON, ni dans l'état OFF et permet aussi d'exprimer qu'à partir du moment où on a mis la variable x à 1, c'est à dire que l'on a éteint la machine, alors on ne la rallume plus avant l'arrivée de la nouvelle tâche. En effet, comme on est dans un problème de minimisation, à partir du moment où x vaut 1, il n'est pas nécessaire d'augmenter la valeur de t_i pour satisfaire la contrainte et on ne le fera donc pas.
On peut largement remarquer la similarité avec le programme linéaire du problème de la rentabilité des skis. Ce problème d'économie d'énergie pour une machine à 2 états peut alors se résoudre directement exactement de la même manière que celui sur la rentabilité des skis. On se demande donc si on pourrait utiliser les approches simples ou l'approche Primal-Dual de la même manière qu'avec le problème de la rentabilité des skis pour résoudre ce problème sur une machine avec plus d'états.

3.2 Pour une machine à 3 états

3.2.1 Définition du problème et algorithmes de résolution

Soit une machine présentant trois états : ON, VEI (état de veille) et OFF. Rester dans l'état ON coûte A_1 à chaque unité de temps, rester en VEI coûte A_2 à chaque unité de temps et rester en OFF coûte A_3 à chaque unité de temps (on a $A_1 > A_2 > A_3$). Par contre, lorsqu'une tâche arrive, il ne coûte rien de l'exécuter si on

est dans l'état ON, il coûte C_1 de l'exécuter si on était dans l'état VEI (coût du passage de VEI à ON), et il coûte C_2 de l'exécuter si on était dans l'état OFF (coût du passage de OFF à ON). On note que l'on ne peut pas passer d'un état "inférieur" vers un état "supérieur" (l'état ON étant supérieur à l'état VEI étant lui-même supérieur à l'état OFF) avant l'arrivée de la tâche puisque cela aurait pour conséquence d'augmenter le coût (on aurait $A_{sup} - A_{inf} > 0$ à payer en plus puisque la machine serait dans l'état supérieur de coût unitaire A_{sup} à la place de rester dans l'état inférieur de coût unitaire A_{inf}) et ce n'est pas souhaitable. Le problème est donc d'économiser l'énergie entre deux arrivées de tâches.

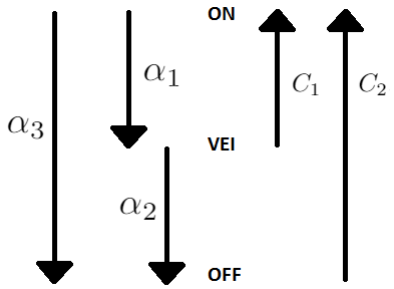

FIGURE 2 – Coût des transitions entre états

Ce problème serait simple si on connaissait les dates d'arrivées des tâches. Plaçons nous dans le cas où une tâche vient de finir et la prochaine tâche arrive dans N unités de temps. Un algorithme déterministe optimal serait alors le suivant : si on devait attendre moins de $\frac{C_1}{A_1 - A_2}$ unités de temps, on laisserait la machine dans l'état ON pour un coût de $N.A_1 \leq \frac{C_1}{A_1 - A_2}.A_1$, si on devait attendre entre $\frac{C_1}{A_1 - A_2}$ et $\frac{C_2 - C_1}{A_2 - A_3}$ unités de temps, on passerait directement à l'état VEI jusqu'à l'arrivée de la prochaine tâche pour un coût de $N.A_2 + C_1 \leq \frac{C_2 - C_1}{A_2 - A_3}.A_2 + C_1$, et si on devait attendre plus de $\frac{C_2 - C_1}{A_2 - A_3}$ unités de temps, on passerait directement à l'état OFF jusqu'à l'arrivée de la prochaine tâche pour un coût de $N.A_3 + C_2 \geq \frac{C_2 - C_1}{A_2 - A_3}.A_3 + C_2$.
Dans le cas en ligne où l'on n'aurait aucune information concernant la durée séparant les deux tâches, l'algorithme LEA présenté dans le document [5] suggère de rester dans l'état ON jusqu'au temps $\frac{C_1}{A_1 - A_2}$ puis de passer à l'état VEI et d'y rester jusqu'au temps $\frac{C_2 - C_1}{A_2 - A_3}$ puis de passer dans l'état OFF et d'y rester jusqu'à l'arrivée de la tâche. Cet algorithme est 2-compétitif.

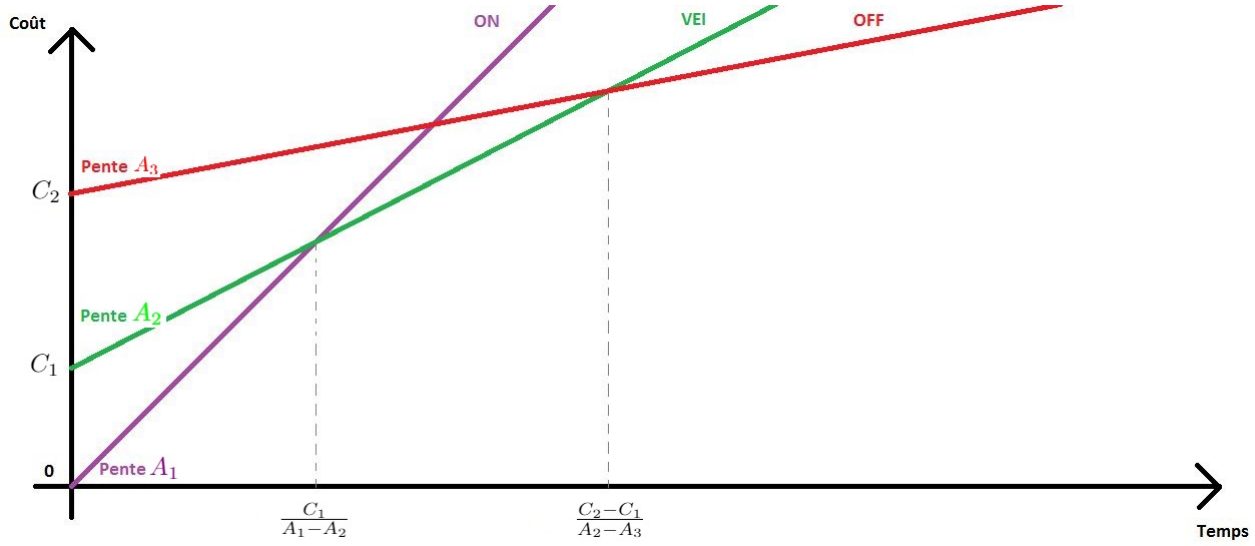

FIGURE 3 – Consommation en fonction de l'état au cours du temps

Dans le cas où l'on possède une approximation du nombre d'unités de temps entre deux tâches N_{pred}, on pourrait, de la même manière qu'avec le problème de la rentabilité des skis, suivre aveuglement les prédictions. On aurait alors l'algorithme suivant :

Algorithme aveugle avec prédictions

Entrée : N_{pred}
Si $N_{pred} < \frac{C_1}{A_1 - A_2}$, alors :
    Rester dans l'état ON jusqu'à l'arrivée d'une nouvelle tâche pour un coût de $A_1.N$
Si $\frac{C_1}{A_1 - A_2} \leq N_{pred} < \frac{C_2 - C_1}{A_2 - A_3}$, alors :
    Passer dans l'état VEI dès le premier instant pour un coût de $A_2.N + C_1$
Sinon :
    Passer dans l'état OFF dès le premier instant pour un coût de $A_3.N + C_2$

Malgré sa simplicité, cet algorithme est consistant puisque lorsque les prédictions sont exactes, il retourne la solution optimale. Cependant, cet algorithme n'est pas robuste. En effet, si la prédiction est très mauvaise, le coût obtenu sera très loin de l'optimal (le ratio de compétitivité sera très mauvais).
On pourrait donc imaginer, à partir de cet algorithme simple, un algorithme qui serait à la fois consistant et robuste en introduisant un hyperparamètre λ (qui est, comme précédemment, le paramètre de robustesse). On aurait alors l'algorithme suivant :

Algorithme robuste et consistant avec prédictions

Entrée : λ, N_{pred}
Si $N_{pred} < \frac{C_1}{A_1 - A_2}$, alors :
    Passer dans l'état VEI au temps $\frac{C_1}{\lambda(A_1 - A_2)}$ puis dans l'état OFF au temps $\frac{C_2 - C_1}{\lambda(A_2 - A_3)}$
Si $\frac{C_1}{A_1 - A_2} \leq N_{pred} < \frac{C_2 - C_1}{A_2 - A_3}$, alors :
    Passer dans l'état VEI au temps $\lambda\frac{C_1}{A_1 - A_2}$ puis passer dans l'état OFF au temps $\frac{C_2 - C_1}{\lambda(A_2 - A_3)}$
Sinon :
    Passer dans l'état VEI au temps $\lambda\frac{C_1}{A_1 - A_2}$ puis dans l'état OFF au temps $\lambda\frac{C_2 - C_1}{A_2 - A_3}$

Cette formulation traduit bien une confiance totale en la prédiction si λ vaut zéro et une méfiance grandissante lorsqu'il se rapproche de un. En effet, lorsque λ est nul, c'est à dire lorsque les prédictions sont supposées excellentes, l'algorithme ci-dessus est équivalent à l'algorithme aveugle : si $N_{pred} < \frac{C_1}{A_1 - A_2}$ alors ne jamais passer dans l'état VEI ni dans l'état OFF ; si $\frac{C_1}{A_1 - A_2} \leq N_{pred} < \frac{C_2 - C_1}{A_2 - A_3}$ alors passer dans l'état VEI au temps 0 puis ne jamais passer dans l'état OFF ; et sinon passer dans l'état VEI puis OFF au temps 0 (donc simplement passer directement à l'état OFF). De même, lorsque λ vaut 1, c'est à dire lorsque les prédictions sont supposées totalement incorrectes, l'algorithme reste plutôt bon et reprends les caractéristiques de l'algorithme déterministe optimal énoncé plus haut.
Dans la version en ligne de ce problème, on ne connaît pas à l'avance le nombre d'unités de temps séparant deux tâches. On doit donc prendre une décision à chaque nouveau pas de temps qui arrive sans remettre en cause les précédentes décisions. On peut ainsi établir un algorithme optimal déterministe qui décidera de rester dans l'état ON pendant les $\frac{C_1}{A_1 - A_2} - 1$ premiers instants, de passer dans l'état VEI si l'on atteint le pas de temps $\frac{C_1}{A_1 - A_2}$, et de passer dans l'état OFF si l'on atteint le pas de temps $\frac{C_2 - C_1}{A_2 - A_3}$. Ces choix seront optimaux si finalement le nombre de pas de temps N n'excède pas $\frac{C_1}{A_1 - A_2}$. Si ce n'est pas le cas, soit le nombre de pas de temps N n'excède pas $\frac{C_2 - C_1}{A_2 - A_3}$ et dans ce cas le choix optimal serait de passer dans l'état VEI dès le premier pas de temps et d'y rester jusqu'à l'arrivée de la tâche pour un coût de $A_2.N + C_1$ alors que notre algorithme renverrait une solution de coût $A_1(\frac{C_1}{A_1 - A_2} - 1) + A_2.(N - \frac{C_1}{A_1 - A_2} + 1) + C_1 = A_1\frac{C_1}{A_1 - A_2} - A_1 + A_2.N - A_2.\frac{C_1}{A_1 - A_2} + A_2. + C_1 = 2C_1 + A_2.(N + 1) - A_1$, ce qui est moins bien si $C_1 + A_2 > A_1$, soit le nombre de pas de temps N excède $\frac{C_2 - C_1}{A_2 - A_3}$ et dans ce cas le choix optimal serait de passer dans l'état OFF dès le premier pas de temps et d'y rester jusqu'à l'arrivée de la tâche pour un coût de $A_3.N + C_2$ alors que notre algorithme renverrait une solution de coût $A_1(\frac{C_1}{A_1 - A_2} - 1) + A_2.(\frac{C_2 - C_1}{A_2 - A_3} - \frac{C_1}{A_1 - A_2}) + A_3.(N - \frac{C_2 - C_1}{A_2 - A_3} + 1) + C_2 = A_1\frac{C_1}{A_1 - A_2} - A_1 + A_2.\frac{C_2 - C_1}{A_2 - A_3} - A_2.\frac{C_1}{A_1 - A_2} + A_3.N - A_3.\frac{C_2 - C_1}{A_2 - A_3} + A_3. + C_2 = C_1 - A_1 + (C_2 - C_1) + A_3.N + A_3 + C_2 = 2C_2 - A_1 + A_3.(N - 1)$, ce qui est moins bien si $C_2 + A_3 > A_1$.
Démonstration : De la même manière que dans le cas à 2 états, nous avons essayer de prouver la robustesse de cet algorithme. Malheureusement, la preuve n'a pas aboutie mais vous pouvez retrouver les pistes de réflexions que l'on a eu en annexe 6.4.
Maintenant que nous avons formulé les deux algorithmes simples dans le cadre de ce problème, intéressons nous

à l'approche Primal-Dual. Nous cherchons donc à résoudre le programme linéaire suivant :

$$Primal : \begin{cases} Min\ C_1.x_{VEI} + C_2.x_{OFF} + \sum_{i=1}^{N}(A_1.ON_i + A_2.VEI_i + A_3.OFF_i) \\ \text{Sous contraintes :} \\ \text{État initial : } ON_0 = 1,\ OFF_0 = 0,\ VEI_0 = 0 \\ \text{Pour chaque unité de temps i (de 1 à N), } ON_i + VEI_i + OFF_i = 1 \\ \text{Pour chaque unité de temps i (de 1 à N), } x_{OFF} \geq OFF_i \\ \text{Pour chaque unité de temps i (de 1 à N), } x_{VEI} \geq VEI_i - x_{OFF} \\ \text{Pour chaque unité de temps i (de 0 à N-1) et pour tout j} > \text{i (de i+1 à N), } ON_i \geq ON_j \\ \text{Pour chaque unité de temps i (de 0 à N-1) et pour tout j} > \text{i (de i+1 à N), } OFF_i \leq OFF_j \\ x_{VEI} \geq 0, x_{OFF} \geq 0, \text{ et } ON_i \geq 0, VEI_i \geq 0, OFF_i \geq 0 \text{ pour chaque unité de temps i} \end{cases} \tag{5}$$

où $x_{VEI} = \begin{cases} 1 & \text{si la machine est dans l'état VEI lorsque la nouvelle tâche arrive} \\ 0 & \text{sinon} \end{cases}$,

$x_{OFF} = \begin{cases} 1 & \text{si la machine est dans l'état OFF lorsque la nouvelle tâche arrive} \\ 0 & \text{sinon} \end{cases}$,

$ON_i = \begin{cases} 1 & \text{si la machine est dans l'état ON au temps i} \\ 0 & \text{sinon} \end{cases}$, $VEI_i = \begin{cases} 1 & \text{si la machine est dans l'état VEI au temps i} \\ 0 & \text{sinon} \end{cases}$,

$OFF_i = \begin{cases} 1 & \text{si la machine est dans l'état OFF au temps i} \\ 0 & \text{sinon} \end{cases}$, et N est le nombre d'unité de temps qui s'écoule entre la fin de la dernière tâche et l'arrivée d'une nouvelle tâche.

Dans la version en ligne de ce problème, c'est-à-dire lorsque l'on ne connaît pas la valeur de N, à chaque nouveau pas de temps, de nouvelles variables ON_i, VEI_i, et OFF_i et leurs contraintes d'intégrité associées s'ajoutent au problème et on doit donc prendre une décision (laisser la machine dans l'état dans lequel elle est ou, si elle était dans l'état ON, choisir de faire une transition vers l'état VEI ou vers l'état OFF, ou encore, si elle était dans l'état VEI, choisir de faire une transition vers l'état OFF) à chaque pas de temps sans remettre en cause les décisions précédentes.

Détails : Les détails sur la formulation de ce programme linéaire et le chemin de pensées pour y parvenir se trouvent en annexe dans la section 6.5.

Dans l'optique d'appliquer l'approche Primal-Dual à ce problème, nous nous intéressons désormais à formuler le dual de ce programme linéaire. Tout d'abord, réécrivons le primal sous forme canonique :

$$Primal : \begin{cases} Min\ C_1.x_{VEI} + C_2.x_{OFF} + \sum_{i=1}^{N}(A_1.ON_i + A_2.VEI_i + A_3.OFF_i) \\ \text{Sous contraintes :} \\ \text{État initial : } ON_0 = 1 & y_1 \\ \qquad\qquad\quad VEI_0 = 0 & y_2 \\ \qquad\qquad\quad OFF_0 = 0 & y_3 \\ \text{Pour chaque unité de temps i (de 1 à N) : } ON_i + VEI_i + OFF_i = 1 & z_i \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad x_{OFF} - OFF_i \geq 0 & w_i \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad x_{VEI} - VEI_i + x_{OFF} \geq 0 & v_i \\ \text{Pour chaque unité de temps i (de 0 à N-1) et pour tout j} > \text{i (de i+1 à N) :} \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad ON_i - ON_j \geq 0 & t_{(i,j)} \\ \qquad\qquad\qquad\qquad\qquad\qquad\quad OFF_j - OFF_i \geq 0 & u_{(i,j)} \\ x_{VEI} \geq 0, x_{OFF} \geq 0, \text{ et } ON_i \geq 0, VEI_i \geq 0, OFF_i \geq 0 \text{ pour chaque unité de temps i} \end{cases} \tag{6}$$

À droite de chaque contrainte est indiqué, en violet, la variable correspondante dans le dual. On note, étant donné le sens de chaque (in)égalité, que les variables duales ont les domaines suivants : $y_1 \in \mathbb{R}, y_2 \in \mathbb{R}, y_3 \in \mathbb{R}$ pour l'état initial ; pour chaque unité de temps i (de 1 à N), $z_i \in \mathbb{R}, w_i \geq 0, v_i \geq 0$; pour chaque unité de temps i (de 0 à N) et pour tout j > i, $t_{(i,j)} \geq 0, u_{(i,j)} \geq 0$. De plus, comme toutes les variables primales sont positives, toutes les contraintes dans le dual sont des inégalités. En appliquant la méthode pour passer du primal au dual,

on obtient le programme linéaire suivant :

$$Dual : \begin{cases} Max\ y_1 + \sum_{i=1}^{N} z_i \\ \text{Sous contraintes :} \\ \text{Variable } x_{VEI} \text{ du Primal : } \sum_{i=1}^{N} v_i \leq C_1 \\ \text{Variable } x_{OFF} \text{ du Primal : } \sum_{i=1}^{N} w_i + \sum_{i=1}^{N} v_i \leq C_2 \\ \text{Variable } ON_0 \text{ du Primal : } \sum_{j=1}^{N} t_{(0,j)} + y_1 \leq A_1 \\ \text{Variable } ON_i \text{ (i entre 1 et N-1) du Primal :} \\ \text{Pour chaque unité de temps i (de 1 à N-1) : } z_i + \sum_{j=i+1}^{N} t_{(i,j)} - \sum_{k=0}^{i-1} t_{(k,i)} \leq A_1 \\ \text{Variable } ON_N \text{ du Primal : } z_i - \sum_{k=0}^{N-1} t_{(k,i)} \leq A_1 \\ \text{Variable } VEI_0 \text{ du Primal : } y_2 \leq A_2 \\ \text{Variable } VEI_i \text{ (i entre 1 et N) du Primal :} \\ \text{Pour chaque unité de temps i (de 1 à N) : } z_i - v_i \leq A_2 \\ \text{Variable } OFF_0 \text{ du Primal : } y_3 - \sum_{j=1}^{N} u_{(0,j)} \leq A_3 \\ \text{Variable } OFF_i \text{ (i entre 1 et N-1) du Primal :} \\ \text{Pour chaque unité de temps i (de 1 à N-1) : } z_i - w_i - \sum_{j=i+1}^{N} u_{(i,j)} + \sum_{k=0}^{i-1} u_{(k,i)} \leq A_3 \\ \text{Variable } OFF_N \text{ du Primal : } z_i - w_i + \sum_{k=0}^{N-1} u_{(k,i)} \leq A_3 \\ y_1 \in \mathbb{R}, y_2 \in \mathbb{R}, y_3 \in \mathbb{R} \\ \text{Pour chaque unité de temps i (de 1 à N) : } z_i \in \mathbb{R}, w_i \geq 0, v_i \geq 0 \\ \text{Pour chaque unité de temps i (de 0 à N) et pour tout j } > \text{ i : } t_{(i,j)} \geq 0, u_{(i,j)} \geq 0 \end{cases} \tag{7}$$

Expliquons, en particulier, le passage de la contrainte $ON_i - ON_j \geq 0$ (pour chaque unité de temps i (de 0 à N-1) et pour tout j>i (de i+1 à N)) du primal vers les contraintes du dual. Pour ce faire, considérons temporairement (pour simplifier) que cette contrainte est la seule du primal et prenons par exemple le cas N = 2. La contrainte primale se réécrit donc : $ON_i - ON_j \geq 0$ (pour chaque unité de temps i (de 0 à 1) et pour tout j>i (de 1 à 2)), soit les contraintes suivantes :

$$\begin{cases} ON_0 - ON_1 \geq 0 & t_{(0,1)} \\ ON_0 - ON_2 \geq 0 & t_{(0,2)} \\ ON_1 - ON_2 \geq 0 & t_{(1,2)} \end{cases} \tag{8}$$

Les variables duales associées sont respectivement $t_{(0,1)}$, $t_{(0,2)}$, et $t_{(1,2)}$ comme indiqué en violet. On peut donc déduire les contraintes duales concernant chaque variable primale en fonction de leur apparition dans chaque contrainte primale. La contrainte duale concernant la variable ON_0 aura ainsi pour membre de gauche $t_{(0,1)} + t_{(0,2)}$ puisque la variable ON_0 apparaît une unique fois dans les deux contraintes désignées par les variables duales $t_{(0,1)}$, et $t_{(0,2)}$. On raisonne de la même manière pour les autres variables primales. La contrainte duale concernant la variable ON_1 aura pour membre de gauche $-t_{(0,1)} + t_{(1,2)}$ et celle concernant la variable ON_2 aura pour membre de gauche $-t_{(0,2)} - t_{(1,2)}$. Plus généralement, on voit bien grâce à cet exemple pourquoi on a bien la formule utilisée dans les contraintes plus haut, c'est à dire $\sum_{j=i+1}^{N} t_{(i,j)} - \sum_{k=0}^{i-1} t_{(k,i)}$ pour ON_i. Le même raisonnement est effectué pour obtenir la formule $-\sum_{j=i+1}^{N} u_{(i,j)} + \sum_{k=0}^{i-1} u_{(k,i)}$ concernant la variable primale OFF_i.
Maintenant que nous avons exprimé les programmes linéaires primal et dual, nous essayons de mettre en place une approche Primal-Dual comme suit :

Algorithme "Primal-Dual" sans prédictions

Initialisation : $ON_j \leftarrow 0$, $VEI_j \leftarrow 0$, $OFF_j \leftarrow 0$ $\forall j$
$ON_0 \leftarrow 1$, $VEI_0 \leftarrow 0$, $OFF_0 \leftarrow 0$
$c_1 \leftarrow A_1$, $c_2 \leftarrow A_2$

Pour chaque nouvelle unité de temps j :
    Si la machine était dans l'état ON au temps précédent (c'est-à-dire si $ON_{j-1} > 0$) :
        Transférer $\frac{1}{c_1} ON_{j-1}$ unités de l'état ON vers l'état VEI
    Sinon :
        Si la machine était dans l'état VEI au temps précédent (c'est-à-dire si $VEI_{j-1} > 0$) :
            Transférer $\frac{1}{c_2} VEI_{j-1}$ unités de l'état VEI vers l'état OFF
    Si $ON_j + VEI_j + OFF_j < 1$:
        $OFF_j \leftarrow 1 - (ON_j + VEI_j + OFF_j)$

Cet algorithme a été inspiré de l'algorithme Primal-Dual pour le problème de la rentabilité des skis. On note que l'on ne retrouve pas clairement d'élément des programmes linéaires établis précédemment car nous ne sommes pas parvenu à déduire un algorithme de ces PLs. On notera tout de même les points positifs de l'algorithme : celui-ci respecte la sommation des variables d'états à 1 pour chaque unité de temps ainsi que la croissance des états en transférant des unités peu à peu vers un état inférieur et jamais vers un état supérieur. Cependant, cet algorithme suggère (de par le caractère non entier des variables) que la machine peut être dans deux états à la fois, ce qui est impossible selon la définition du problème mais nous avons opté pour cette simplification. De plus, les constantes c_1 et c_2 sont totalement arbitraires et ont besoin d'être réfléchi d'avantage afin de conserver les optimisations déterministes détaillées par la figure 3. On aimerait, en effet, que la machine ne soit plus du tout dans l'état ON (mais dans un état inférieur) au temps $\frac{C_1}{A_1 - A_2}$ et que la machine soit complètement dans l'état OFF à partir du temps $\frac{C_2 - C_1}{A_2 - A_3}$. La première contrainte va donc définir la constante c_1. On aimerait en effet qu'au bout de $\frac{C_1}{A_1 - A_2}$ mises à jour, la variable ON vaille 0, c'est à dire que $(1 - \frac{1}{c_1})^{\frac{C_1}{A_1 - A_2}} = 1$ puisque pour chaque pas de temps i, ON_i prend la valeur de $(1 - \frac{1}{c_1}) ON_{i-1}$. On obtient donc $1 - \frac{1}{c_1} = 1$, soit $c_1 = 1$ mais cette constante ne va pas nous arranger puisqu'elle représente un passage de l'état ON vers l'état VEI au pas de temps 2 alors qu'il aurait peut-être été plus judicieux de rester dans l'état ON plus longtemps. On peut raisonner de la même façon pour c_2. Nous pouvons donc remettre en cause la formule de mise à jour des variables. En effet, si les constantes c_1 et c_2 sont différentes de 1, comme on vient de le voir, la modification $x_j \leftarrow (1 - \frac{1}{c}) x_{j-1}$ (où x est l'état ON ou VEI selon l'état dans lequel était l'algorithme à l'instant $j - 1$ et c correspond à la constante c_1 ou c_2 selon les cas) ne converge pas (ou très lentement) vers 0 ce qui est problématique vis à vis de ce que l'on souhaite faire. On voudrait en effet, à chaque pas de temps, transmettre des unités de l'état courant (c'est-à-dire l'état le plus haut dont la variable n'est pas à 0) vers l'état directement inférieur. Pour ce faire, on peut penser à une formule de mise à jour de la forme $x_j \leftarrow x_{j-1} - \frac{1}{c}$ (où x est l'état ON ou VEI selon l'état dans lequel était l'algorithme à l'instant $j - 1$ et c correspond à la constante c_1 ou c_2 selon les cas). On peut ainsi adapter les constantes c_1 et c_2 afin de conserver les optimisations déterministes détaillées par la figure 3. On aimerait en effet que la machine ne soit plus du tout dans l'état ON (mais dans un état inférieur) au temps $\frac{C_1}{A_1 - A_2}$ et que la machine soit complètement dans l'état OFF à partir du temps $\frac{C_2 - C_1}{A_2 - A_3}$. La première contrainte va donc définir la constante c_1. On aimerait en effet qu'au bout de $\frac{C_1}{A_1 - A_2}$ mises à jour, la variable ON vaille 0, c'est à dire que $\frac{C_1}{A_1 - A_2} \frac{1}{c_1} = 1$ puisqu'on retire $\frac{1}{c_1}$ unités à l'état ON à chaque pas de temps. On obtient donc $c_1 = \frac{C_1}{A_1 - A_2}$. La deuxième contrainte va, elle, définir la constante c_2. La première contrainte permet de dire que la variable VEI va valoir 1 au plus tard au temps $\frac{C_1}{A_1 - A_2}$. Puisqu'il faut qu'elle vaille 0 au plus tard au temps $\frac{C_2 - C_1}{A_2 - A_3}$, alors on aimerait qu'au bout de $(\frac{C_2 - C_1}{A_2 - A_3} - \frac{C_1}{A_1 - A_2})$ mises à jour, la variable VEI vaille 0, c'est à dire que $(\frac{C_2 - C_1}{A_2 - A_3} - \frac{C_1}{A_1 - A_2}) \frac{1}{c_2} = 1$, soit $c_2 = (\frac{C_2 - C_1}{A_2 - A_3} - \frac{C_1}{A_1 - A_2})$. Cependant, l'algorithme précédemment décrit s'éloigne d'avantage de l'approche Primal-Dual et n'est en fait qu'une version flottante de l'algorithme déterministe optimal sans prédiction. Il n'apporte donc pas beaucoup d'information.
Plaçons nous désormais dans le cas où l'on possède une approximation du nombre d'instants entre deux tâches N_{pred} (toujours dans la version en ligne du problème évidemment). De la même manière que dans le cas précédent, à chaque nouvel instant j, la machine doit décider de rester dans l'état actuel ou d'aller vers un état inférieur. Pour ce faire, l'algorithme va se servir de la prédiction N_{pred} afin d'ajuster le taux de variation des variables. En particulier, lorsque $N_{pred} < \frac{C_1}{A_1 - A_2}$, l'algorithme va utiliser des constantes c_1 et c_2 élevées afin de passer lentement de l'état ON vers l'état VEI puis de l'étata VEI vers l'état OFF, puisque les prédictions suggèrent de rester dans l'état ON jusqu'à l'arrivée de la prochaine tâche. Notons que l'algorithme aurait pu prendre un paramètre supplémentaire λ qui permettrait de quantifier cette augmentation. Cette dernière s'effec-

tuera, en effet, de manière plus ou moins drastique en fonction du paramètre de robustesse λ. Intuitivement, λ indique notre confiance en la prédiction, plus λ sera petit plus on aura confiance en la prédiction. L'algorithme est le suivant :

Algorithme "Primal-Dual" avec prédictions

Entrée : N_{pred}
Initialisation : $ON_j \leftarrow 0$, $VEI_j \leftarrow 0$, $OFF_j \leftarrow 0$ $\forall j$
$ON_0 \leftarrow 1$, $VEI_0 \leftarrow 0$, $OFF_0 \leftarrow 0$

Si $N_{pred} < \frac{C_1}{A_1 - A_2}$, alors :
    /*Les prédictions suggèrent de rester dans l'état ON
    $c_1 \leftarrow ELEVE$, $c_2 \leftarrow ELEVE$
Sinon :
    Si $N_{pred} < \frac{C_2 - C_1}{A_2 - A_3}$, alors :
        /*Les prédictions suggèrent de passer dans l'état VEI dès le premier instant et d'y rester
        $c_1 \leftarrow FAIBLE$, $c_2 \leftarrow ELEVE$
    Sinon :
        /*Les prédictions suggèrent de passer dans l'état OFF dès le premier instant et d'y rester
        $c_1 \leftarrow FAIBLE$, $c_2 \leftarrow FAIBLE$

Pour chaque nouvelle unité de temps j :
    Si la machine était dans l'état ON au temps précédent (c'est-à-dire si $ON_{j-1} > 0$) :
        Transférer $\frac{1}{c_1} ON_{j-1}$ unités de l'état ON vers l'état VEI
    Sinon :
        Si la machine était dans l'état VEI au temps précédent (c'est-à-dire si $VEI_{j-1} > 0$) :
            Transférer $\frac{1}{c_2} VEI_{j-1}$ unités de l'état VEI vers l'état OFF
    Si $ON_j + VEI_j + OFF_j < 1$:
        $OFF_j \leftarrow 1 - (ON_j + VEI_j + OFF_j)$

On note que l'on a bien un passage lent vers l'état inférieur lorsque la constante est grande et un passage plus rapide lorsque la constante est petite. En effet, comme leurs noms l'indique, on a $FAIBLE < ELEVE$.
Cet algorithme a été, comme le précédent, inspiré de l'algorithme Primal-Dual pour le problème de la rentabilité des skis. On note que l'on ne retrouve pas clairement d'élément des programmes linéaires établis précédemment car nous ne sommes pas parvenu à déduire un algorithme de ces PLs. On notera tout de même que l'algorithme respecte la sommation des variables d'états à 1 pour chaque unité de temps ainsi que la croissance des états en transférant des unités peu à peu vers un état inférieur et jamais vers un état supérieur. Cependant, cet algorithme suggère (de par le caractère non entier des variables) que la machine peut être dans deux états à la fois, ce qui est impossible selon la définition du problème mais nous avons opté pour cette simplification. De plus, les constantes $FAIBLE$ et $ELEVE$ (fixées respectivement à 2 et 100 pour effectuer les simulations) sont totalement arbitraires et ont besoin d'être réfléchi d'avantage afin de conserver les optimisations détaillées précédemment (bien que dans tous les cas, le problème de convergence persiste). De la même manière qu'avec le cas sans prédictions, nous pouvons donc remettre en cause la formule de mise à jour des variables. En effet, la modification $x_j \leftarrow (1 - \frac{1}{c}) x_{j-1}$ (où x est l'état ON ou VEI selon l'état dans lequel était l'algorithme à l'instant $j-1$ et c correspond à la constante c_1 ou c_2 selon les cas) ne converge pas (ou très lentement) vers 0 ce qui est problématique vis à vis de ce que l'on souhaite faire. On voudrait en effet, à chaque pas de temps, transmettre des unités de l'état courant (c'est-à-dire l'état le plus haut dont la variable n'est pas à 0) vers l'état directement inférieur. Pour ce faire, on peut penser à une formule de mise à jour de la forme $x_j \leftarrow x_{j-1} - \frac{1}{c}$ (où x est l'état ON ou VEI selon l'état dans lequel était l'algorithme à l'instant $j-1$ et c correspond à la constante c_1 ou c_2 selon les cas). On peut ainsi adapter les constantes c_1 et c_2 afin de conserver les optimisations déterministes détaillées par la figure 3 mais nous retrouvons les mêmes problèmes que précédemment et finissons par obtenir simplement une version flottante de l'algorithme déterministe optimal sans prédiction. Il n'apporte donc pas beaucoup d'information.

3.2.2 Simulations

Comparons désormais les performances des algorithmes simples. Pour ce faire, nous avons effectué des simulations sur 15 000 exemples. Nous avons échantillonné l'axe des X en 100 parties égales. Pour le choix

des données, nous avons pris A_1 à 5, A_2 à 3, A_3 à 0 (on a donc bien $A_1 > A_2 > A_3$), C_1 à 10, C_2 à 25 et le nombre maximum d'instant entre les tâches à 30. Nous tirons donc uniformément N entre 1 et 30 puis pour la prédiction nous prenons le N précédemment obtenu auquel nous ajoutons un entier aléatoire entre 0 et 30-N et nous retirons un nombre aléatoire entre 0 et N-1. Ainsi nous obtenons un Npred entre 1 et 30 qui est probablement proche du N.

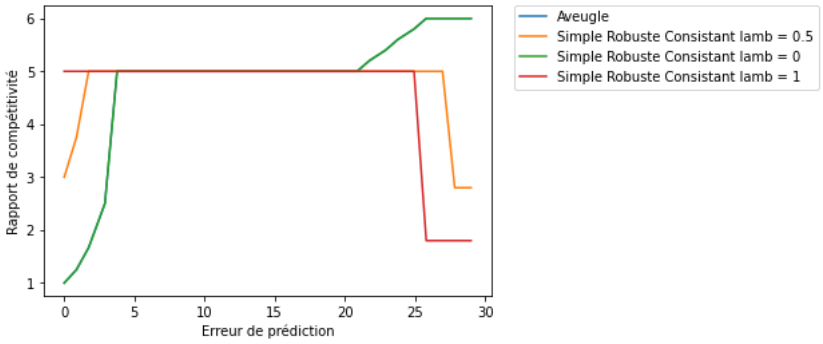

FIGURE 4 – Comparaison des algorithmes en fonction de l'erreur de prédiction

Notons tout d'abord que la courbe bleue est exactement superposée à la courbe rouge, ce qui est logique puisque dans les deux cas l'algorithme associé suit aveuglement la prédiction. Avec ce graphique, nous voyons donc bien d'intérêt de l'approche simple robuste et consistante par rapport à l'approche aveugle. En effet, lorsque les prédictions sont imparfaites (erreur de prédiction supérieure à 0), l'approche aveugle devient très vite aussi mauvaise que l'approche simple robuste et consistante (avec un rapport de compétitivité à 5) et devient même encore plus mauvais lorsque les erreurs deviennent plus élevées. L'algorithme simple robuste et consistant semble avoir un rapport de compétitivité borné à 5 pour un λ supérieur à 0.5. Notons également que plus l'erreur de prédiction est élevée, meilleur est l'algorithme robuste et consistant avec un λ élevé (avec de meilleurs résultats pour $\lambda = 1$), ce qui est logique puisqu'un λ élevé signifie une méfiance envers la prédiction et induit donc une robustesse vis à vis de l'erreur au dépend de la consistance. Par contre, lorsque l'erreur de prédiction est faible, meilleur est l'algorithme robuste et consistant avec un λ faible (avec de meilleurs résultats pour $\lambda = 0$), ce qui est logique puisqu'un λ faible signifie une confiance envers la prédiction et induit donc une consistance au dépend de la robustesse.
Les simulations pour l'approche "Primal-Dual" sont accessibles en annexe dans la section 6.6.

3.3 Pour une machine à k états

3.3.1 Définition du problème et algorithmes de résolution

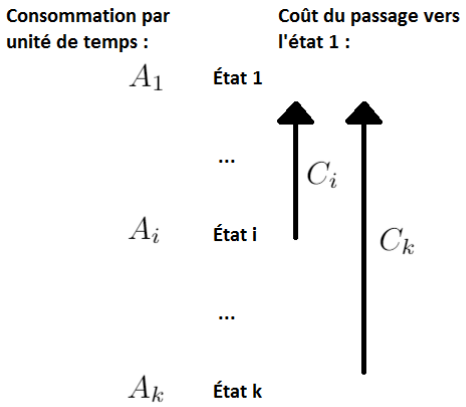

FIGURE 5 – Coût des transitions entre états

Soit une machine présentant k états, l'état 1 étant l'état ON et l'état k, l'état OFF. Rester dans l'état i coûte A_i à chaque unité de temps (on a $A_i > A_j \ \forall i < j$). Lorsqu'une nouvelle tâche arrive, il coûte C_i de l'exécuter si la machine est dans l'état i à ce moment là (c'est en fait le coût de passage de l'état courant i vers l'état 1). On a, en particulier, $C_1 = 0$ puisque la machine est déjà dans l'état 1 et peut donc directement exécuter la tâche. On note que l'on ne peut pas passer d'un état "inférieur" vers un état "supérieur" (l'état 1 étant l'état

le plus supérieur, plus généralement l'état i est supérieur à l'état j pour tout i < j) avant l'arrivée de la tâche puisque cela aurait pour conséquence d'augmenter le coût (on aurait $A_{sup} - A_{inf} > 0$ à payer en plus puisque la machine serait dans l'état supérieur de coût unitaire A_{sup} à la place de rester dans l'état inférieur de coût unitaire A_{inf}) et ce n'est pas souhaitable. Le problème est donc d'économiser l'énergie entre deux arrivées de tâches.
Ce problème serait simple si on connaissait les dates d'arrivées des tâches. Plaçons nous dans le cas où une tâche vient de finir et la prochaine tâche arrive dans N unités de temps. Un algorithme déterministe optimal serait alors le suivant : si on devait attendre moins de $\frac{C_1}{A_1 - A_2}$ unités de temps, on laisserait la machine dans l'état 1 pour un coût de $N.A_1$, si on devait attendre entre $\frac{C_1}{A_1 - A_2}$ et $\frac{C_2 - C_1}{A_2 - A_3}$ unités de temps, on passerait directement à l'état 2 jusqu'à l'arrivée de la prochaine tâche pour un coût de $N.A_2 + C_2$, etc. Plus généralement, si on devait attendre entre $\frac{C_i - C_{i-1}}{A_{i-1} - A_i}$ et $\frac{C_{i+1} - C_i}{A_i - A_{i+1}}$ unités de temps, on passerait directement à l'état i jusqu'à l'arrivée de la prochaine tâche pour un coût de $N.A_i + C_i$. On rappelle que l'on a posé $C_1 = 0$ puisqu'il n'y a aucun coût supplémentaire si la machine reste dans l'état 1 jusqu'à l'arrivée de la prochaine tâche.
En effet, comme on peut le voir sur la figure ci-dessous, chaque état i a une consommation dans le temps représentable par une droite d'équation $A_i x + C_i$ (où x est la variable temporelle), puisque le coût pour rester dans l'état i est de A_i par unité de temps et le coût pour revenir à l'état 1 lorsque la tâche arrive est de C_i. On peut donc facilement calculer l'abscisse du point d'intersection de deux droites (disons, sans perte de généralité, les droites représentant les consommations pour les états i et j respectivement) en résolvant l'équation $A_i x + C_i = A_j x + C_j$, soit $(A_i - A_j)x = C_j - C_i$, soit $x = \frac{C_j - C_i}{A_i - A_j}$. On voit facilement qu'avant ce point il est préférable de rester dans l'état i et qu'au delà de celui-ci, il vaut mieux passer dans l'état j.

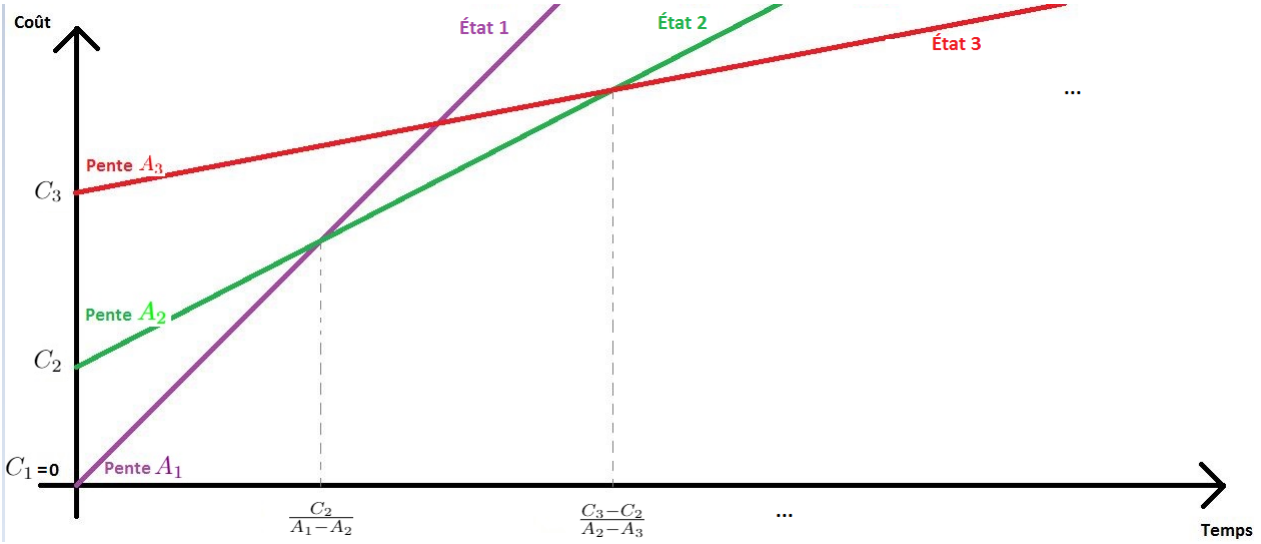

FIGURE 6 – Consommation en fonction de l'état au cours du temps

Dans le cas en ligne où l'on n'aurait aucune information concernant la durée séparant les deux tâches, sur la base de l'algorithme LEA présenté dans l'article [5], un algorithme optimal serait de passer dans l'état i au temps $\frac{C_i - C_{i-1}}{A_{i-1} - A_i}$ pour tout i.
Dans le cas où l'on possède une approximation du nombre d'unités de temps entre deux tâches N_{pred}, on pourrait, de la même manière qu'avec le problème de la rentabilité des skis, suivre aveuglement les prédictions. On aurait alors l'algorithme suivant :

Algorithme aveugle avec prédictions

Entrée : N_{pred}

Si $N_{pred} < \frac{C_2}{A_1 - A_2}$, alors :
Rester dans l'état 1 jusqu'à l'arrivée d'une nouvelle tâche pour un coût de $A_1.N$

Pour tout i de 2 à k-1 :
Si $\frac{C_i - C_{i-1}}{A_{i-1} - A_i} \leq N_{pred} < \frac{C_{i+1} - C_i}{A_i - A_{i+1}}$, alors :
Passer dans l'état i dès le premier instant pour un coût de $A_i.N + C_i$

Sinon :
Passer dans l'état k dès le premier instant pour un coût de $A_k.N + C_k$

Malgré sa simplicité, cet algorithme est consistant puisque lorsque les prédictions sont exactes, il retourne la solution optimale. Cependant, cet algorithme n'est pas robuste. En effet, si la prédiction est très mauvaise, le coût obtenu sera très loin de l'optimal (le ratio de compétitivité sera très mauvais).
On pourrait donc imaginer, à partir de cet algorithme simple, un algorithme qui serait à la fois consistant et robuste en introduisant un hyperparamètre λ (qui est, comme précédemment, le paramètre de robustesse). On aurait alors l'algorithme suivant :

Algorithme robuste et consistant avec prédictions

Entrée : λ, N_{pred}
Si $N_{pred} < \frac{C_2}{A_1 - A_2}$, alors :
Passer dans l'état 2 au temps $\frac{C_2}{\lambda(A_1 - A_2)}$ puis dans l'état 3 au temps $\frac{C_3 - C_2}{\lambda(A_2 - A_3)}$, puis, pour tout i, dans l'état i au temps $\frac{C_i - C_{i-1}}{\lambda(A_{i-1} - A_i)}$

Pour tout i de 2 à k-1 :
Si $\frac{C_i - C_{i-1}}{A_{i-1} - A_i} \leq N_{pred} < \frac{C_{i+1} - C_i}{A_i - A_{i+1}}$, alors :
Pour tout $l < i$, passer dans l'état l au temps $\lambda \frac{C_l - C_{l-1}}{A_{l-1} - A_l}$, puis, pour tout $j \geq i$, dans l'état j au temps $\frac{C_j - C_{j-1}}{\lambda(A_{j-1} - A_j)}$

Sinon :
Pour tout $l \leq k$, passer dans l'état l au temps $\lambda \frac{C_l - C_{l-1}}{A_{l-1} - A_l}$

Cette formulation traduit bien une confiance totale en la prédiction si λ vaut zéro et une méfiance grandissante lorsqu'il se rapproche de un. En effet, lorsque λ est nul, c'est à dire lorsque les prédictions sont supposées excellentes, l'algorithme ci-dessus est équivalent à l'algorithme aveugle : si $N_{pred} < \frac{C_2}{A_1 - A_2}$ alors ne jamais passer dans un état $i > 1$; si $\frac{C_i - C_{i-1}}{A_{i-1} - A_i} \leq N_{pred} < \frac{C_{i+1} - C_i}{A_i - A_{i+1}}$ alors passer dans l'état $i - 1$ au temps 0 puis ne jamais passer dans un état $j > i$. De même, lorsque λ vaut 1, c'est à dire lorsque les prédictions sont supposées totalement incorrectes, l'algorithme reste plutôt bon et reprends les caractéristiques de l'algorithme déterministe optimal énoncé plus haut.
Dans la version en ligne de ce problème, on ne connaît pas à l'avance le nombre d'unités de temps séparant deux tâches. On doit donc prendre une décision à chaque nouveau pas de temps qui arrive sans remettre en cause les précédentes décisions. On peut ainsi établir un algorithme optimal déterministe qui décidera de rester dans l'état 1 pendant les $\frac{C_2}{A_1 - A_2}$ premiers instants, puis pour tout i allant de 2 à k, de passer dans l'état i si l'on atteint le pas de temps $\frac{C_i - C_{i-1}}{A_{i-1} - A_i}$. Ces choix seront optimaux si finalement le nombre de pas de temps N n'excède pas $\frac{C_2}{A_1 - A_2}$.
Maintenant que nous avons formulé les deux algorithmes simples dans le cadre de ce problème, intéressons nous

à l'approche Primal-Dual. Nous cherchons donc à résoudre le programme linéaire suivant :

$$Primal : \begin{cases} Min \sum_{i=2}^{k} C_i.x_i + \sum_{j=1}^{N}(\sum_{i=1}^{k} A_i.et_{(i,j)}) \\ \text{Sous contraintes :} \\ \text{État initial : } et_{(1,0)} = 1,\ \sum_{i=2}^{k} et_{(i,0)} = 0 \\ \text{Pour chaque unité de temps j (de 1 à N), } \sum_{i=1}^{k} et_{(i,j)} = 1 \\ \text{Pour chaque unité de temps j (de 1 à N) et pour tout i (de 1 à k), } x_i \geq et_{(i,j)} - \sum_{l=i+1}^{k} x_l \\ \text{Pour tout i (de 1 à k), pour chaque unité de temps j (de 1 à N-1), et pour tout l} > \text{j (de j+1 à N),} \\ et_{(i,j)} \geq \sum_{r=1}^{i-1} et_{(r,l)} - \sum_{h=1}^{i-1} et_{(h,j)} \\ \text{Pour tout i (de 1 à k), et pour chaque unité de temps j (de 0 à N), } x_i \geq 0, \text{ et } et_{(i,j)} \geq 0 \end{cases} \tag{9}$$

où $x_i = \begin{cases} 1 & \text{si la machine est dans l'état i lorsque la nouvelle tâche arrive} \\ 0 & \text{sinon} \end{cases}$,

$et_{(i,j)} = \begin{cases} 1 & \text{si la machine est dans l'état i au temps j} \\ 0 & \text{sinon} \end{cases}$, et N est le nombre d'unité de temps qui s'écoule entre la fin de la dernière tâche et l'arrivée d'une nouvelle tâche.

Dans la version en ligne de ce problème, c'est-à-dire lorsque l'on ne connaît pas la valeur de N, à chaque nouveau pas de temps j, de nouvelles variables $et_{(i,j)}$ et leurs contraintes d'intégrité associées s'ajoutent au problème et on doit donc prendre une décision (laisser la machine dans l'état dans lequel elle est ou, si elle n'était dans l'état k, choisir de faire une transition vers un état inférieur) à chaque pas de temps sans remettre en cause les décisions précédentes.

Détails : Les détails sur la formulation de ce programme linéaire et le chemin de pensées pour y parvenir se trouvent en annexe dans la section 6.7.

Dans l'optique d'appliquer l'approche Primal-Dual à ce problème, nous nous intéressons désormais à formuler le dual de ce programme linéaire. Tout d'abord, réécrivons le primal sous forme canonique :

$$Primal : \begin{cases} Min \sum_{i=2}^{k} C_i.x_i + \sum_{j=1}^{N}(\sum_{i=1}^{k} A_i.et_{(i,j)}) \\ \text{Sous contraintes :} \\ \text{État initial : } et_{(1,0)} = 1 & y_1 \\ \qquad\qquad \sum_{i=2}^{k} et_{(i,0)} = 0 & y_2 \\ \text{Pour chaque unité de temps j (de 1 à N) : } \sum_{i=1}^{k} et_{(i,j)} = 1 & z_i \\ \text{Pour chaque unité de temps j (de 1 à N) et pour tout i (de 1 à k) :} \\ x_i - et_{(i,j)} + \sum_{l=i+1}^{k} x_l \geq 0 & w_{(i,j)} \\ \text{Pour chaque unité de temps j (de 1 à N-1), pour tout i (de 1 à k) et pour tout l} > \text{j (de j+1 à N) :} \\ et_{(i,j)} - \sum_{r=1}^{i-1} et_{(r,l)} + \sum_{h=1}^{i-1} et_{(h,j)} \geq 0 & v_{(i,j,l)} \\ x_i \geq 0, \text{ et } et_{(i,j)} \geq 0 \text{ pour tout i allant de 1 à k, et pour chaque unité de temps j (de 0 à N)} \end{cases} \tag{10}$$

À droite de chaque contrainte est indiqué, en violet, la variable correspondante dans le dual. On note, étant donné le sens de chaque (in)égalité, que les variables duales ont les domaines suivants : $y_1 \in \mathbb{R}, y_2 \in \mathbb{R}$ pour l'état initial ; pour chaque unité de temps j (de 1 à N), $z_j \in \mathbb{R}$; pour chaque unité de temps j (de 1 à N) et pour tout état i (de 1 à k), $w_{(i,j)} \geq 0$; et pour chaque unité de temps j (de 1 à N), pour tout état i (de 1 à k) et pour tout $l > j$, $v_{(i,j,l)} \geq 0$. De plus, comme toutes les variables primales sont positives, toutes les contraintes dans le dual sont des inégalités. En appliquant la méthode pour passer du primal au dual, on obtient le programme

linéaire suivant :

$$Dual : \begin{cases} Max\ y_1 + \sum_{j=1}^{N} z_j \\ \text{Sous contraintes :} \\ \text{Variable } x_i \text{ (i entre 1 et k) du Primal :} \\ \text{Pour chaque état i (de 1 à k) : } \sum_{j=1}^{N} w_{(i,j)} + \sum_{j=1}^{N} \sum_{l=1}^{i-1} w_{(l,j)} \leq C_i \\ \text{Variable } et_{(1,0)} \text{ du Primal : } y_1 \leq A_1 \\ \text{Variable } et_{(i,0)} \text{ (i entre 2 et k) du Primal : Pour chaque état i (de 2 à k) : } y_2 \leq A_i \\ \text{Variable } et_{(i,j)} \text{ (i entre 1 et k, j entre 0 et N) du Primal :} \\ \text{Pour chaque unité de temps j (de 0 à N) et pour chaque état i (de 1 à k) :} \\ z_i - \sum_{j=1}^{N} w_{(i,j)} + \sum_{l=j+1}^{N} v_{(i,j,l)} - \sum_{h=1}^{j-1} \sum_{r=i+1}^{k} v_{(r,h,j)} + \sum_{h=j+1}^{N} \sum_{r=i+1}^{k} v_{(r,j,h)} \leq A_i \\ y_1 \in \mathbb{R}, y_2 \in \mathbb{R} \\ \text{Pour chaque unité de temps j (de 1 à N) : } z_j \in \mathbb{R} \\ \text{Pour chaque unité de temps j (de 1 à N) et pour tout état i (de 1 à k) : } w_{(i,j)} \geq 0 \\ \text{Pour chaque unité de temps j (de 1 à N), pour tout état i (de 1 à k) et pour tout } l > j : v_{(i,j,l)} \geq 0 \end{cases} \tag{11}$$

Expliquons, en particulier, le passage de la contrainte $et_{(i,j)} - \sum_{r=1}^{i-1} et_{(r,l)} + \sum_{h=1}^{i-1} et_{(h,j)} \geq 0$ (pour chaque unité de temps j (de 1 à N-1), pour tout i (de 1 à k) et pour tout l > j (de j+1 à N)) du primal vers les contraintes du dual. Pour ce faire, considérons temporairement (pour simplifier) que cette contrainte est la seule du primal et prenons par exemple le cas N = 2 et k = 3. La contrainte primal se réécrit donc : $et_{(i,j)} - \sum_{r=1}^{i-1} et_{(r,l)} + \sum_{h=1}^{i-1} et_{(h,j)} \geq 0$ (pour chaque unité de temps j (de 1 à 1), pour tout i (de 1 à 3) et pour tout l > j (de j+1 à 2)). On peut réécrire cela sous la forme des trois contraintes suivantes (pour j = 1, l = 2 et i variant de 1 à 3) :

$$\begin{cases} et_{(1,1)} \geq 0 & v_{(1,1,2)} \\ et_{(2,1)} - et_{(1,2)} + et_{(1,1)} \geq 0 & v_{(2,1,2)} \\ et_{(3,1)} - et_{(1,2)} - et_{(2,2)} + et_{(1,1)} + et_{(2,1)} \geq 0 & v_{(3,1,2)} \end{cases} \tag{12}$$

Les variables duales associées sont respectivement $v_{(1,1,2)}$, $v_{(2,1,2)}$, et $v_{(3,1,2)}$ comme indiqué en violet. On peut donc déduire les contraintes duales concernant chaque variable primale en fonction de leur apparition dans chaque contrainte primale. La contrainte duale concernant la variable $et_{(1,1)}$ aura ainsi pour membre de gauche $v_{(1,1,2)} + v_{(2,1,2)} + v_{(3,1,2)}$ puisque la variable $et_{(1,1)}$ apparaît une unique fois dans chacune des trois contraintes désignées par les variables duales $v_{(1,1,2)}$, $v_{(2,1,2)}$, et $v_{(3,1,2)}$. On raisonne de la même manière pour les autres variables primales. La contrainte duale concernant la variable $et_{(1,2)}$ aura pour membre de gauche $-v_{(2,1,2)} - v_{(3,1,2)}$, celle concernant la variable $et_{(2,1)}$ aura pour membre de gauche $v_{(2,1,2)} + v_{(3,1,2)}$, celle concernant la variable $et_{(3,1)}$ aura pour membre de gauche $v_{(3,1,2)}$, et celle concernant la variable $et_{(2,2)}$ aura pour membre de gauche $v_{(3,1,2)}$. On retrouve bien les mêmes expressions que celles données par la formule $\sum_{l=j+1}^{N} v_{(i,j,l)} - \sum_{h=1}^{j-1} \sum_{r=i+1}^{k} v_{(r,h,j)} + \sum_{h=j+1}^{N} \sum_{r=i+1}^{k} v_{(r,j,h)}$ pour $et_{(i,j)}$.
Maintenant que nous avons exprimé les programmes linéaires primal et dual, nous essayons de mettre en place une approche Primal-Dual comme suit :

Algorithme "Primal-Dual" sans prédictions

Initialisation : $et_{(i,j)} \leftarrow 0\ \forall i, \forall j$
$et_{(1,0)} \leftarrow 1$, $et_{(i,0)} \leftarrow 0\ \forall i$
$c_i \leftarrow A_i\ \forall i$

Pour chaque nouvelle unité de temps j :
    Pour chaque état i :
        Si la machine était dans l'état i au temps précédent (c'est-à-dire si $et_{(i,j-1)} > 0$) :
            Transférer $\frac{1}{c_i} et_{(i,j-1)}$ unités de l'état i vers l'état $i+1$ et sortir de la boucle
    Si $\sum_{i=1}^{k} et_{(i,j)} < 1$:
        $et_{(k,j)} \leftarrow 1 - \sum_{i=1}^{k-1} et_{(i,j)}$

Cet algorithme a été inspiré de l'algorithme Primal-Dual pour le problème de la rentabilité des skis. On note que l'on ne retrouve pas clairement d'élément des programmes linéaires établis précédemment car nous ne sommes

pas parvenu à déduire un algorithme de ces PLs. On notera tout de même que l'algorithme respecte la sommation des variables d'états à 1 pour chaque unité de temps ainsi que la croissance des états en transférant des unités peu à peu vers un état inférieur et jamais vers un état supérieur. Cependant, cet algorithme suggère (de par le caractère non entier des variables) que la machine peut être dans deux états à la fois, ce qui est impossible selon la définition du problème mais nous avons opté pour cette simplification. De plus, les constantes c_i sont totalement arbitraires et ont besoin d'être réfléchi d'avantage afin de conserver les optimisations déterministes détaillées par la figure 6. On aimerait, en effet, que la machine ne soit plus du tout dans l'état i (mais dans un état inférieur) au temps $\frac{C_{i+1}-C_i}{A_i-A_{i+1}}$. Réfléchissons tout d'abord à la contrainte pour l'état 1, qui va définir la constante c_1. On aimerait, en effet, qu'au bout de $\frac{C_1}{A_1-A_2}$ mises à jour, la variable $et(1,t)$ vaille 0 (où t est le pas de temps considéré), c'est à dire que $(1-\frac{1}{c_1})^{\frac{C_1}{A_1-A_2}} = 1$ puisque pour chaque pas de temps t, $et_{(1,t)}$ prend la valeur de $(1-\frac{1}{c_1})et_{(1,t-1)}$. On obtient donc $1-\frac{1}{c_1} = 1$, soit $c_1 = 1$ mais cette constante ne va pas nous arranger puisqu'elle représente un passage de l'état 1 vers l'état 2 au pas de temps 2 alors qu'il aurait peut-être été plus judicieux de rester dans l'état 1 plus longtemps. On peut raisonner de la même façon pour tout les c_i. Nous pouvons donc remettre en cause la formule de mise à jour des variables. En effet, si les constantes c_i sont différentes de 1, comme on vient de le voir, la modification $et_{(i,j)} \leftarrow (1-\frac{1}{c_i})et_{(i,j-1)}$ (où i est le numéro de l'état dans lequel était l'algorithme à l'instant $j-1$) ne converge pas (ou très lentement) vers 0 ce qui est problématique vis à vis de ce que l'on souhaite faire. On voudrait en effet, à chaque pas de temps, transmettre des unités de l'état courant (c'est-à-dire l'état le plus haut dont la variable n'est pas à 0) vers l'état directement inférieur. Pour ce faire, on peut penser à une formule de mise à jour de la forme $et_{(i,j)} \leftarrow et_{(i,j-1)} - \frac{1}{c_i}$ (où i est le numéro de l'état dans lequel était l'algorithme à l'instant $j-1$). On peut ainsi adapter les constantes c_i afin de conserver les optimisations déterministes détaillées par la figure 6. On aimerait, en effet, que la machine ne soit plus du tout dans l'état i (mais dans un état inférieur) au temps $\frac{C_{i+1}-C_i}{A_i-A_{i+1}}$. Réfléchissons tout d'abord à la contrainte pour l'état 1, qui va définir la constante c_1. On aimerait en effet qu'au bout de $\frac{C_1}{A_1-A_2}$ mises à jour, la variable $et_{(1,t)}$ vaille 0 (où t est le pas de temps considéré), c'est à dire que $\frac{C_1}{A_1-A_2}\frac{1}{c_1} = 1$ puisqu'on retire $\frac{1}{c_1}$ unités à l'état 1 à chaque pas de temps. On obtient donc $c_1 = \frac{C_1}{A_1-A_2}$. Les contraintes suivantes vont, elles, définir les constantes c_i pour tout $i > 1$. Réfléchissons donc à la contrainte pour l'état i, qui va définir la constante c_i. La contrainte $i-1$ permet de dire que la variable $et_{(i,t)}$ va valoir 1 au plus tard au temps $\frac{C_i-C_{i-1}}{A_{i-1}-A_i}$ (temps auquel la machine ne sera plus du tout dans l'état $i-1$). Puisqu'il faut qu'elle vaille 0 au plus tard au temps $\frac{C_{i+1}-C_i}{A_i-A_{i+1}}$, alors on aimerait qu'au bout de $(\frac{C_{i+1}-C_i}{A_i-A_{i+1}} - \frac{C_i-C_{i-1}}{A_{i-1}-A_i})$ mises à jour, la variable $et_{(i,t)}$ vaille 0, c'est à dire que $(\frac{C_{i+1}-C_i}{A_i-A_{i+1}} - \frac{C_i-C_{i-1}}{A_{i-1}-A_i})\frac{1}{c_i} = 1$, soit $c_i = (\frac{C_{i+1}-C_i}{A_i-A_{i+1}} - \frac{C_i-C_{i-1}}{A_{i-1}-A_i})$. Cependant, l'algorithme précédemment décrit s'éloigne d'avantage de l'approche Primal-Dual et n'est en fait qu'une version flottante de l'algorithme déterministe optimal sans prédiction. Il n'apporte donc pas beaucoup d'information.

Plaçons nous désormais dans le cas où l'on possède une approximation du nombre d'instants entre deux tâches N_{pred} (toujours dans la version en ligne du problème évidemment). De la même manière que dans le cas précédent, à chaque nouvel instant j, la machine doit décider de rester dans l'état actuel ou d'aller vers un état inférieur. Pour ce faire, l'algorithme va se servir de la prédiction N_{pred} afin d'ajuster le taux de variation des variables. En particulier, lorsque $N_{pred} < \frac{C_1}{A_1-A_2}$, l'algorithme va utiliser des constantes c_1 et c_2 élevées afin de passer lentement de l'état ON vers un état inférieur, puisque les prédictions suggèrent de rester dans l'état ON jusqu'à l'arrivée de la prochaine tâche. Notons que l'algorithme aurait pu prendre un paramètre supplémentaire λ qui permettrait de quantifier cette augmentation. Cette dernière s'effectuera, en effet, de manière plus ou moins drastique en fonction du paramètre de robustesse λ. Intuitivement, λ indique notre confiance en la prédiction, plus λ sera petit plus on aura confiance en la prédiction. L'algorithme est le suivant :

Algorithme "Primal-Dual" avec prédictions

Entrée : N_{pred}

Initialisation : $et_{(i,j)} \leftarrow 0 \ \forall i, \forall j$
$et_{(1,0)} \leftarrow 1$, $et_{(i,0)} \leftarrow 0 \ \forall i$

Pour chaque état i :
Si $N_{pred} < \frac{C_i - C_{i-1}}{A_{i-1} - A_i}$, alors :
/*Les prédictions suggèrent pour tout $l < i$, de passer dans l'état l rapidement, puis pour tout $r \geq i$, de passer dans l'état r moins vite
$c_l \leftarrow FAIBLE \ \forall l < i$, $c_r \leftarrow ELEVE \ \forall r \geq i$ et sortir de la boucle
Si $N_{pred} \geq \frac{C_{k-1} - C_{k-2}}{A_{k-2} - A_{k-1}}$, alors :
/*Les prédictions suggèrent pour tout $l < k$, de passer dans l'état l rapidement
$c_l \leftarrow FAIBLE \ \forall l < k$

Pour chaque nouvelle unité de temps j :
Pour chaque état i :
Si la machine était dans l'état i au temps précédent (c'est-à-dire si $et_{(i,j-1)} > 0$) :
Transférer $\frac{1}{c_i} et_{(i,j-1)}$ unités de l'état i vers l'état $i+1$ et sortir de la boucle
Si $\sum_{i=1}^{k} et_{(i,j)} < 1$:
$et_{(k,j)} \leftarrow 1 - \sum_{i=1}^{k-1} et_{(i,j)}$

On note que l'on a bien un passage lent vers l'état inférieur lorsque la constante est grande et un passage plus rapide lorsque la constante est petite. En effet, comme leurs noms l'indique, on a $FAIBLE < ELEVE$.
Cet algorithme a été, comme le précédent, inspiré de l'algorithme Primal-Dual pour le problème de la rentabilité des skis. On note que l'on ne retrouve pas clairement d'élément des programmes linéaires établis précédemment car nous ne sommes pas parvenu à déduire un algorithme de ces PLs. On notera tout de même que l'algorithme respecte la sommation des variables d'états à 1 pour chaque unité de temps ainsi que la croissance des états en transférant des unités peu à peu vers un état inférieur et jamais vers un état supérieur. Cependant, cet algorithme suggère (de par le caractère non entier des variables) que la machine peut être dans deux états à la fois, ce qui est impossible selon la définition du problème mais nous avons opté pour cette simplification. De plus, les constantes $FAIBLE$ et $ELEVE$ (fixées respectivement à 2 et 100 pour effectuer les simulations) sont totalement arbitraires et ont besoin d'être réfléchi d'avantage afin de conserver les optimisations détaillées précédemment (bien que dans tous les cas, le problème de convergence persiste). De la même manière qu'avec le cas sans prédictions, nous pouvons donc remettre en cause la formule de mise à jour des variables. En effet, la modification $et_{(i,j)} \leftarrow (1 - \frac{1}{c_i}) et_{(i,j-1)}$ (où i est le numéro de l'état dans lequel était l'algorithme à l'instant $j-1$) ne converge pas (ou très lentement) vers 0 ce qui est problématique vis à vis de ce que l'on souhaite faire. On voudrait en effet, à chaque pas de temps, transmettre des unités de l'état courant (c'est-à-dire l'état le plus haut dont la variable n'est pas à 0) vers l'état directement inférieur. Pour ce faire, on peut penser à une formule de mise à jour de la forme $et_{(i,j)} \leftarrow et_{(i,j-1)} - \frac{1}{c_i}$ (où i est le numéro de l'état dans lequel était l'algorithme à l'instant $j-1$). On peut ainsi adapter les constantes c_i afin de conserver les optimisations déterministes détaillées par la figure 6 mais nous retrouvons les mêmes problèmes que précédemment et finissons par obtenir simplement une version flottante de l'algorithme déterministe optimal sans prédiction. Il n'apporte donc pas beaucoup d'information.

3.3.2 Simulations

Comparons désormais les performances des algorithmes simples. Pour ce faire, nous avons effectué des simulations sur $5000k$ exemples (qui est en fonction du nombre d'état k pour plus de représentativité). Nous avons échantillonné l'axe des X en 100 parties égales. Pour le choix des données, nous avons pris A_i à $2(k-i-1)$ pour tout i de 1 à k (on a donc bien $A_1 > A_2 > ... > A_k$) et C_i à $5(i-1)$ pour tout i de 1 à k (on a donc bien $C_1 = 0$ et $C_1 < C_2 < ... < C_k$), et le nombre maximum d'instant entre les tâches à $10 * k$ (qui est en fonction du nombre d'état pour avoir potentiellement le temps de parcourir tous les états, en effet, si il y a toujours le même temps maximum entre deux tâches que le nombre d'état soit grand ou petit, cela poserait problème). Nous tirons donc uniformément N entre 1 et $10 * k$ puis pour la prédiction nous prenons le N précédemment obtenu auquel nous ajoutons un entier aléatoire entre 0 et $10 * k - N$ et nous retirons un nombre aléatoire entre 0 et N-1. Ainsi nous obtenons un Npred entre 1 et $10 * k$ qui est probablement proche du N.

Nous voyons donc bien d'intérêt de l'approche simple robuste et consistante par rapport à l'approche aveugle.

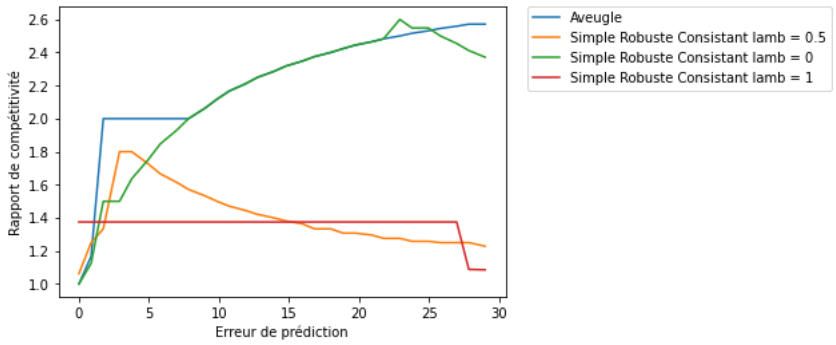

FIGURE 7 – Comparaison des algorithmes en fonction de l'erreur de prédiction pour k=3

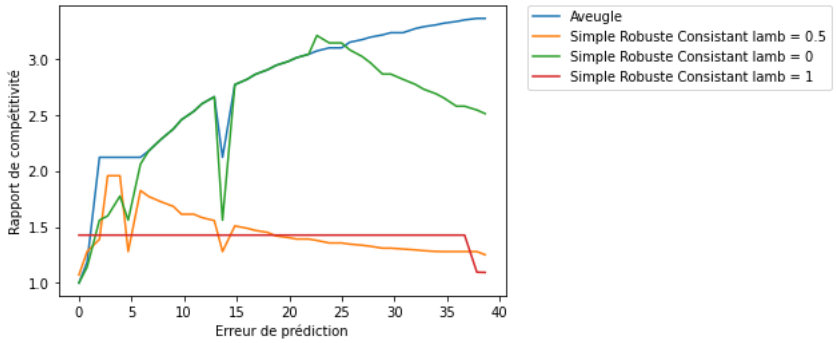

FIGURE 8 – Comparaison des algorithmes en fonction de l'erreur de prédiction pour k=4

En effet, lorsque les prédictions sont imparfaites (erreur de prédiction supérieure à 0), l'approche aveugle devient très vite moins bonne que l'approche simple robuste et consistante avec un $\lambda > 0$ et cela se détériore lorsque les erreurs deviennent plus élevées. L'algorithme simple robuste et consistant semble avoir un rapport de compétitivité borné à 2 pour un λ supérieur à 0.5 et ce rapport semble dépendre de k puisqu'il était de 5 pour $k = 3$. Notons également que plus l'erreur de prédiction est élevée, meilleur est l'algorithme robuste et consistant avec un λ élevé (avec de meilleurs résultats pour $\lambda = 1$), ce qui est logique puisqu'un λ élevé signifie une méfiance envers la prédiction et induit donc une robustesse vis à vis de l'erreur au dépend de la consistance. Par contre, lorsque l'erreur de prédiction est faible, meilleur est l'algorithme robuste et consistant avec un λ faible (avec de meilleurs résultats pour $\lambda = 0$), ce qui est logique puisqu'un λ faible signifie une confiance envers la prédiction et induit donc une consistance au dépend de la robustesse. De plus, on note que les courbes vertes et bleues ne sont pas superposées comme on pourrait s'y attendre et que les simulations obtenues pour $k = 3$ ne sont pas les mêmes que celles obtenues par les algorithmes encodées spécialement pour le cas $k = 3$ (voir section précédente), cela est sûrement dû à l'implémentation de ces algorithmes.
Les simulations pour l'approche "Primal-Dual" sont accessibles en annexe dans la section numéro 6.8.

4 Conclusion

En conclusion, ce stage a permis de généraliser le problème d'économie d'énergie pour une machine à états, de trouver des solutions pour ces problèmes et de mettre en avant, grâce aux simulations pour le problème de la rentabilité des skis, l'avantage de l'approche Primal-Dual par rapport aux approches simples. Nous pourrions donc poursuivre ce travail en cherchant d'autres méthodes pour le problème d'économie d'énergie pour une machine à états qui s'apparenterait plus à une approche Primal-Dual que ce que l'on a fait. En effet, le manque de temps ne nous a pas permis de poursuivre comme on le voudrait sur cette voie.
Je tiens à remercier vivement mon tuteur de stage et professeur, Mr Evripidis Bampis, professeur-chercheur de l'équipe RO du LIP6 de Sorbonne Université, pour son accueil, le prêt de matériel, le temps passé ensemble et le partage de son expertise au quotidien. Grâce aussi à sa confiance et à l'indépendance qui en découle, j'ai pu m'accomplir dans mes missions. Je remercie également toute l'équipe RO pour leur accueil et pour l'accès au bureaux.

5 Bibliographie

[1] N. Buchbinder and J. Naor, « The Design of Competitive Online Algorithms via a Primal-Dual Approach », 2009.

[2] R. Kumar, M. Purohit, and Z. Svitkina, « Improving Online Algorithms via ML Predictions », Google Mountain View, CA, 2018.

[3] E. Bamas, A. Maggiori, and O. Svensson, « The Primal-Dual method for Learning Augmented Algorithms », EPF, Lausanne, Switzerland, oct. 2020.

[4] E. Bamas, A. Maggiori, O. Svensson, and L. Rohwedder, « Learning Augmented Energy Minimization via Speed Scaling », EPF, Switzerland, oct. 2020.

[5] S. Irani, S. Shukla, and R. Gupta, « Online Strategies for Dynamic Power Management in Systems with Multiple Power-Saving States », University of California at Irvine, aug. 2003.

[6] David P. Williamson, and David P. Shmoys, « The Design of Approximation Algorithms », Cambridge University Press, 2011.

[7] S. Im, R. Kumar, M. Qaem, and M. Purohit, « Non-Clairvoyant Scheduling with Predictions », University of California, Merced/Google Research, 2021.

[8] J. Augustine, S. Irani, and C. Swamyy, « Optimal Power-Down Strategies », School of Information and Computer Science, University of California at Irvine, Center for the Mathematics of Information Caltech, Pasadena, CA, aug. 2003.

6 Annexes

6.1 Preuve du théorème sur le coût de l'algorithme Primal-Dual avec prédictions (Ski Rental)

Commençons par prouver la borne de consistance, c'est à dire le premier paramètre du minimum du théorème.
On note tout d'abord qu'après chaque mise à jour, le primal augmente de $1 + \frac{1}{c-1}$. En effet, au jour j-1, le primal est défini comme $B.x + \sum_{i=1}^{j-1} z_i$. Ensuite, au jour j, on met à jour les variables comme suit : $z_j \leftarrow 1 - x$ et $x \leftarrow (1 + \frac{1}{B})x + (\frac{1}{(c-1).B})$. Le nouveau primal s'écrit donc comme $B.[(1 + \frac{1}{B})x + (\frac{1}{(c-1).B})] + \sum_{i=1}^{j-1} z_i + 1 - x$ $= (B + 1)x + (\frac{1}{(c-1)}) + \sum_{i=1}^{j-1} z_i + 1 - x = B.x + \sum_{i=1}^{j-1} z_i + 1 + (\frac{1}{(c-1)})$.
Nous allons désormais faire une distinction de cas en fonction de la valeur de c.
Si $N_{pred} \geq B$, l'algorithme va favoriser l'achat par rapport à la location. Dans ce cas, au plus λB mises à jour sont effectuées avant d'avoir $x \geq 1$. En effet, si on note x_k la valeur de x après k itérations, on a la formule suivante : $x_k \geq \frac{e(\frac{k}{B})-1}{e(\lambda)-1}$. Prouvons cela par récurrence : pour k=0, il est évident que $x \geq \frac{e(0)-1}{e(\lambda)-1} = 0$ par définition du domaine de x. Supposons que le résultat est vrai pour k et montrons qu'il est par conséquent aussi vrai pour l'itération k+1.
$x_{k+1} = (1 + \frac{1}{B})x_k + (\frac{1}{(e(\lambda-1).B)})$
$x_{k+1} \geq (1 + \frac{1}{B})\frac{e(\frac{k}{B})-1}{e(\lambda)-1} + (\frac{1}{(e(\lambda-1).B)})$ selon l'hypothèse de récurrence
$x_{k+1} \geq \frac{(1+\frac{1}{B})(e(\frac{k}{B})-1)+\frac{1}{B}}{e(\lambda)-1}$ en mettant tout au même dénominateur
$x_{k+1} \geq \frac{e(\frac{k}{B})+\frac{1}{B}e(\frac{k}{B})-1-\frac{1}{B}+\frac{1}{B}}{e(\lambda)-1}$ en distribuant la première multiplication
$x_{k+1} \geq \frac{e(\frac{k+1}{B})-1}{e(\lambda)-1}$ puisque $e(\frac{k+1}{B}) = (1 + \frac{1}{B})^{k+1} = (1 + \frac{1}{B})^k e(\frac{k}{B})$
On a donc bien le résultat attendu pour l'hérédité. On peut conclure par récurrence que l'on a bien $x_k \geq \frac{e(\frac{k}{B})-1}{e(\lambda)-1}, \forall k$.
Ainsi, au bout de λB itérations, on aura $x_{\lambda B} \geq \frac{e(\lambda)-1}{e(\lambda)-1} = 1$.
Notons qu'une fois que $x \geq 1$, plus aucune mise à jour n'est nécessaire puisque toute nouvelle contrainte sera déjà vérifiée (les skis sont déjà achetés, on peut donc faire face à tout nouveau jour qui arrive).
De plus, nous avons vu plus haut que chaque mise à jour coûte au plus $1 + \frac{1}{c-1}$, c'est à dire $1 + \frac{1}{e(\lambda)-1}$ dans notre cas de figure, soit $\frac{e(\lambda)}{e(\lambda)-1}$ en mettant tout au même dénominateur, soit $\frac{1}{1-e(-\lambda)}$ en divisant par $e(\lambda)$.
Par conséquent, dans ce cas, l'algorithme Pimal-Dual avec prédictions coûte au plus $\frac{\lambda B}{1-e(-\lambda)} = S(N_{pred}, I)\frac{\lambda}{1-e(-\lambda)}$.
Si $N_{pred} < B$, l'algorithme va favoriser la location par rapport à l'achat. Dans ce cas, chaque mise à jour coûte au plus $1 + \frac{1}{e(\frac{1}{\lambda})-1}$, soit $\frac{e(\frac{1}{\lambda})}{e(\frac{1}{\lambda})-1}$ en mettant tout au même dénominateur, soit $\frac{1}{1-e(-\frac{1}{\lambda})}$ en divisant par $e(\frac{1}{\lambda})$.
De plus, N mises à jour sont faites au maximum pour cet algorithme étant donné qu'il y a N jours et qu'il ne faut qu'une mise à jour maximum par nouveau jour qui arrive.
Par conséquent, dans ce cas, l'algorithme Pimal-Dual avec prédictions coûte au plus $\frac{N}{1-e(-\frac{1}{\lambda})} = S(N_{pred}, I)\frac{1}{1-e(-\frac{1}{\lambda})}$.
Enfin, on sait que $\frac{1}{1-e(-\frac{1}{\lambda})} \leq \frac{\lambda}{1-e(-\lambda)}$, d'où un coût maximal de l'algorithme Pimal-Dual avec prédictions dans ce cas de $S(N_{pred}, I)\frac{\lambda}{1-e(-\lambda)}$.
On a donc bien, dans les deux cas, le coût de l'algorithme borné par $S(N_{pred}, I)\frac{\lambda}{1-e(-\lambda)}$, d'où la borne de consistance.
Prouvons désormais la borne de robustesse, c'est à dire le second paramètre du minimum du théorème.
Tout d'abord, on sait que la solution du dual est réalisable. Essayons maintenant de borner le ratio $\frac{\Delta P}{\Delta D}$. Lorsque l'on effectue de grosses mises à jour (c'est à dire lorsque $N_{pred} \geq B$), on a :

$$\frac{\Delta P}{\Delta D} = \frac{\Delta P}{1} = 1 + \frac{1}{e(\lambda) - 1} = \frac{e(\lambda)}{e(\lambda) - 1} = \frac{1}{1 - e(-\lambda)}$$

Dans le second cas, lorsque l'on effectue de petites mises à jour (c'est à dire lorsque $N_{pred} < B$), on a :

$$\frac{\Delta P}{\Delta D} = \frac{\Delta P}{\lambda} = \frac{1}{\lambda}.(1 + \frac{1}{e(\frac{1}{\lambda}) - 1}) = \frac{1}{\lambda}.\frac{e(\frac{1}{\lambda})}{e(\frac{1}{\lambda}) - 1} = \frac{1}{\lambda}.\frac{1}{1 - e(-\frac{1}{\lambda})} \leq \frac{1}{1 - e(-\lambda)}$$

Par dualité faible, on a donc la borne de robustesse.

6.2 Preuve du ratio de robustesse (Ski Rental)

Commençons par la première borne.
Si $N_{pred} \geq B$, l'algorithme va favoriser l'achat à un jour moins tardif puisque $\lceil \lambda \mathrm{B} \rceil \leq \lceil \frac{B}{\lambda} \rceil$. Dans ce cas, l'algorithme achète les skis au début du jour $\lceil \lambda \mathrm{B} \rceil$ pour un coût de B + $\lceil \lambda \mathrm{B} \rceil$ - 1 si $N \geq \lceil \lambda \mathrm{B} \rceil$ et un coût de N sinon (puisqu'on n'atteint pas le jour d'achat des skis). Le cas où l'on aurait le pire ratio de compétitivité sera lorsque N = $\lceil \lambda \mathrm{B} \rceil$ puisque l'optimum serait de coût $\lceil \lambda B \rceil \leq B$ alors que notre algorithme renverra une solution de coût $B + \lceil \lambda \mathrm{B} \rceil - 1 \leq B + \lambda B \leq \frac{1+\lambda}{\lambda} \lceil \lambda \mathrm{B} \rceil = \frac{1+\lambda}{\lambda}.OPT(I)$.
Si $N_{pred} < B$, l'algorithme va favoriser l'achat à un jour plus tardif pour la même raison que précédemment. Dans ce cas, l'algorithme achète les skis au début du jour $\lceil \frac{B}{\lambda} \rceil$ pour un coût de B + $\lceil \frac{B}{\lambda} \rceil$ - 1 si $N \geq \lceil \frac{B}{\lambda} \rceil$ et un coût de N sinon (puisqu'on n'atteint pas le jour d'achat des skis). Le cas où l'on aurait le pire ratio de compétitivité sera lorsque N = $\lceil \frac{B}{\lambda} \rceil$ puisque l'optimum serait de coût $B \leq \lceil \frac{B}{\lambda} \rceil$ alors que notre algorithme renverra une solution de coût $B + \lceil \frac{B}{\lambda} \rceil - 1 \leq B + \frac{B}{\lambda} \leq \frac{1+\lambda}{\lambda}.OPT(I)$.
Pour montrer la seconde borne, considérons les deux cas suivants :
Si $N_{pred} \geq B$, alors pour tout N tel que N < $\lceil \lambda \mathrm{B} \rceil$, l'algorithme donne une solution de même coût que l'optimum, c'est à dire de coût N. Par contre, lorsque N $\geq \lceil \lambda \mathrm{B} \rceil$, l'algorithme retourne une solution de coût $B + \lceil \lambda \mathrm{B} \rceil - 1 \leq (1+\lambda)B \leq (1+\lambda)(OPT(I) + \eta)$. La seconde inégalité découle du fait que l'optimum a un coût de B si $N \geq B$ et un coût de N tel que $B \leq N_{pred} \leq OPT(I) + \eta$ sinon.
Si $N_{pred} < B$, alors pour tout N tel que $N \leq B$, l'algorithme donne une solution de même coût que l'optimum, c'est à dire de coût N. De la même manière, pour tout N entre B et $\lceil \frac{B}{\lambda} \rceil$, l'algorithme retourne une solution de coût $N \leq N_{pred} + \eta < B + \eta = OPT(I) + \eta$. Finalement, pour tout N tel que N $\geq \lceil \frac{B}{\lambda} \rceil$, notons que $\eta = N - N_{pred} > \frac{B}{\lambda} - B = (1-\lambda)\frac{B}{\lambda}$. L'algorithme retourne alors une solution de coût $B + \lceil \frac{B}{\lambda} \rceil - 1 \leq B + \frac{B}{\lambda} < B + \frac{1}{1-\lambda}\eta = OPT(I) + \frac{1}{1-\lambda}\eta$. Nous obtenons ainsi que le coût renvoyé par notre algorithme est inférieur ou égal à $(1+\lambda).OPT(I) + \frac{1}{1-\lambda}\eta$, ce qui termine la preuve.

6.3 Preuve du ratio de robustesse (Machine à 2 états)

Commençons par la première borne.
Si $N_{pred} \geq \frac{C}{A}$, l'algorithme va favoriser le passage à l'état OFF à un instant moins tardif puisque $\lceil \lambda \frac{C}{A} \rceil \leq \lceil \frac{C}{\lambda A} \rceil$. Dans ce cas, la machine va passer dans l'état OFF à l'instant $\lceil \lambda \frac{C}{A} \rceil$ pour un coût de $C + A(\lceil \lambda \frac{C}{A} \rceil - 1)$ si $N \geq \lceil \lambda \frac{C}{A} \rceil$ et un coût de $A.N$ sinon (puisqu'on n'atteint pas l'instant de passage à l'état OFF, on reste donc dans l'état ON pendant les N instants). Le cas où l'on aurait le pire ratio de compétitivité sera lorsque N = $\lceil \lambda \frac{C}{A} \rceil$ puisque l'optimum serait de rester dans l'état ON jusqu'à l'arrivée de la tâche pour un coût de $A\lceil \lambda \frac{C}{A} \rceil \leq A.\frac{C}{A} = C$ alors que notre algorithme renverra une solution de coût $C + A(\lceil \lambda \frac{C}{A} \rceil - 1) \leq C + A(\lambda \frac{C}{A}) = C + \lambda C \leq \frac{1+\lambda}{\lambda} \lceil \lambda C \rceil \leq \frac{1+\lambda}{\lambda} A \lceil \lambda \frac{C}{A} \rceil = \frac{1+\lambda}{\lambda}.OPT(I)$.
Si $N_{pred} < \frac{C}{A}$, l'algorithme va favoriser le passage à l'état OFF à un instant plus tardif pour la même raison que précédemment. Dans ce cas, la machine va passer dans l'état OFF à l'instant $\lceil \frac{C}{\lambda A} \rceil$ pour un coût de $C + A(\lceil \frac{C}{\lambda A} \rceil - 1)$ si $N \geq \lceil \frac{C}{\lambda A} \rceil$ et un coût de $A.N$ sinon (puisqu'on n'atteint pas l'instant de passage à l'état OFF, on reste donc dans l'état ON pendant les N instants). Le cas où l'on aurait le pire ratio de compétitivité sera lorsque N = $\lceil \frac{C}{\lambda A} \rceil$ puisque l'optimum serait de passer dans l'état OFF dès le premier instant pour un coût de $C \leq A \lceil \frac{C}{\lambda A} \rceil$ alors que notre algorithme renverra une solution de coût $C + A(\lceil \frac{C}{\lambda A} \rceil - 1) \leq C + A(\frac{C}{\lambda A}) \leq C + \frac{C}{\lambda} \leq \frac{1+\lambda}{\lambda} C = \frac{1+\lambda}{\lambda}.OPT(I)$.
Pour montrer la seconde borne, considérons les deux cas suivants :
Si $N_{pred} \geq \frac{C}{A}$, alors pour tout N tel que N < $\lceil \lambda \frac{C}{A} \rceil$, l'algorithme donne une solution de même coût que l'optimum, c'est à dire de coût $A.N$. Par contre, lorsque N $\geq \lceil \lambda \frac{C}{A} \rceil$, l'algorithme retourne une solution de coût $C + A(\lceil \lambda \frac{C}{A} \rceil - 1) \leq C + A(\lambda \frac{C}{A}) = C + \lambda C = (1+\lambda)C \leq (1+\lambda)(OPT(I) + A.\eta)$. La seconde inégalité découle du fait que l'optimum a un coût de C si $N \geq \frac{C}{A}$ et un coût de $A.N$ tel que $\frac{C}{A} \leq N_{pred} \leq OPT(I) + \eta$ sinon.
Si $N_{pred} < \frac{C}{A}$, alors pour tout N tel que $N \leq \frac{C}{A}$, l'algorithme donne une solution de même coût que l'optimum, c'est à dire de coût $A.N$. De la même manière, pour tout N entre $\frac{C}{A}$ et $\lceil \frac{C}{\lambda A} \rceil$, l'algorithme retourne une solution de coût $A.N \leq A.(N_{pred} + \eta) < A(\frac{C}{A} + \eta) = OPT(I) + A.\eta$. Finalement, pour tout N tel que N $\geq \lceil \frac{C}{\lambda A} \rceil$, notons que $\eta = N - N_{pred} > \frac{C}{\lambda A} - \frac{C}{A} = (1-\lambda)\frac{C}{\lambda A}$. L'algorithme retourne alors une solution de coût $C + A(\lceil \frac{C}{\lambda A} \rceil - 1) \leq C + A(\frac{C}{\lambda A}) \leq C + \frac{C}{\lambda} < C + \frac{A}{1-\lambda}\eta = OPT(I) + \frac{A}{1-\lambda}\eta$. Nous obtenons ainsi que le coût renvoyé par notre algorithme est inférieur ou égal à $(1+\lambda).OPT(I) + \frac{A}{1-\lambda}\eta$, ce qui termine la preuve.

6.4 Ébauche de preuve du ratio de robustesse (Machine à 3 états)

Afin d'aller dans la direction de la preuve, nous avons les résultats suivants :
Si $N_{pred} < \frac{C_1}{A_1-A_2}$, l'algorithme va favoriser le passage aux états VEI et OFF à un instant plus tardif puisque $\lambda\frac{C}{A} \leq \frac{C}{\lambda A}$. Le cas où l'on aurait le pire ratio de compétitivité sera lorsque N = $\frac{C_1}{\lambda(A_1-A_2)}$ ou lorsque N = $\frac{C_2-C_1}{\lambda(A_2-A_3)}$. En effet, lorsque N = $\frac{C_1}{\lambda(A_1-A_2)}$, si $A_2 + C_1 > A_1$, l'optimum serait de rester dans l'état ON jusqu'à l'arrivée de la tâche pour un coût de $A_1\frac{C_1}{\lambda(A_1-A_2)}$ alors que notre algorithme renverra une solution de coût $A_1(\frac{C_1}{\lambda(A_1-A_2)} - 1) + A_2 + C_1 = OPT(I) + A_1 - A_2 - C_1$, ce qui est moins bien si $A_2 + C_1 > A_1$. De l'autre côté, lorsque N = $\frac{C_2-C_1}{\lambda(A_2-A_3)}$, si $A_2 + C_1 > A_3 + C_2$, l'optimum serait de passer dans l'état VEI au temps $\frac{C_2-C_1}{\lambda(A_2-A_3)}$ et d'y rester jusqu'à l'arrivée de la tâche pour un coût de $A_1(\frac{C_1}{\lambda(A_1-A_2)} - 1) + A_2(\frac{C_2-C_1}{\lambda(A_2-A_3)} - \frac{C_1}{\lambda(A_1-A_2)} + 1) + C_1$ alors que notre algorithme renverra une solution de coût $A_1(\frac{C_1}{\lambda(A_1-A_2)} - 1) + A_2(\frac{C_2-C_1}{\lambda(A_2-A_3)} - \frac{C_1}{\lambda(A_1-A_2)}) + A_3 + C_2 = OPT(I) - A_2 - C_1 + A_3 + C_2$, ce qui est moins bien si $A_2 + C_1 > A_3 + C_2$.
Si $\frac{C_1}{A_1-A_2} \leq N_{pred} < \frac{C_2-C_1}{A_2-A_3}$, l'algorithme va favoriser le passage à l'état VEI à un instant moins tardif et celui à l'état OFF à un instant plus tardif puisque $\lambda\frac{C}{A} \leq \frac{C}{\lambda A}$. Le cas où l'on aurait le pire ratio de compétitivité sera lorsque N = $\lambda\frac{C_1}{A_1-A_2}$ ou lorsque N = $\frac{C_2-C_1}{\lambda(A_2-A_3)}$. En effet, lorsque N = $\lambda\frac{C_1}{A_1-A_2}$, si $A_2+C_1 > A_1$, l'optimum serait de rester dans l'état ON jusqu'à l'arrivée de la tâche pour un coût de $A_1\lambda\frac{C_1}{A_1-A_2}$ alors que notre algorithme renverra une solution de coût $A_1(\lambda\frac{C_1}{A_1-A_2} - 1) + A_2 + C_1 = OPT(I) + A_1 - A_2 - C_1$, ce qui est moins bien si $A_2 + C_1 > A_1$. De l'autre côté, lorsque N = $\frac{C_2-C_1}{\lambda(A_2-A_3)}$, si $A_2+C_1 > A_3+C_2$, l'optimum serait de passer dans l'état VEI au temps $\frac{C_2-C_1}{\lambda(A_2-A_3)}$ et d'y rester jusqu'à l'arrivée de la tâche pour un coût de $A_1(\lambda\frac{C_1}{A_1-A_2} - 1) + A_2(\frac{C_2-C_1}{\lambda(A_2-A_3)} - \lambda\frac{C_1}{A_1-A_2} + 1) + C_1$ alors que notre algorithme renverra une solution de coût $A_1(\lambda\frac{C_1}{A_1-A_2} - 1) + A_2(\frac{C_2-C_1}{\lambda(A_2-A_3)} - \lambda\frac{C_1}{A_1-A_2}) + A_3 + C_2 = OPT(I) - A_2 - C_1 + A_3 + C_2$, ce qui est moins bien si $A_2 + C_1 > A_3 + C_2$.
Si $N_{pred} \geq \frac{C_2-C_1}{A_2-A_3}$, l'algorithme va favoriser le passage aux états VEI et OFF à un instant moins tardif puisque $\lambda\frac{C}{A} \leq \frac{C}{\lambda A}$. Le cas où l'on aurait le pire ratio de compétitivité sera lorsque N = $\lambda\frac{C_1}{A_1-A_2}$ ou lorsque N = $\lambda\frac{C_2-C_1}{A_2-A_3}$. En effet, lorsque N = $\lambda\frac{C_1}{A_1-A_2}$, si $A_2 + C_1 > A_1$, l'optimum serait de rester dans l'état ON jusqu'à l'arrivée de la tâche pour un coût de $A_1\lambda\frac{C_1}{A_1-A_2}$ alors que notre algorithme renverra une solution de coût $A_1(\lambda\frac{C_1}{A_1-A_2} - 1) + A_2 + C_1 = OPT(I) + A_1 - A_2 - C_1$, ce qui est moins bien si $A_2 + C_1 > A_1$. De l'autre côté, lorsque N = $\lambda\frac{C_2-C_1}{A_2-A_3}$, si $A_2 + C_1 > A_3 + C_2$, l'optimum serait de passer dans l'état VEI au temps $\lambda\frac{C_2-C_1}{A_2-A_3}$ et d'y rester jusqu'à l'arrivée de la tâche pour un coût de $A_1(\lambda\frac{C_1}{A_1-A_2} - 1) + A_2(\lambda\frac{C_2-C_1}{A_2-A_3} - \lambda\frac{C_1}{A_1-A_2} + 1) + C_1$ alors que notre algorithme renverra une solution de coût $A_1(\lambda\frac{C_1}{A_1-A_2} - 1) + A_2(\lambda\frac{C_2-C_1}{A_2-A_3} - \lambda\frac{C_1}{A_1-A_2}) + A_3 + C_2 = OPT(I) - A_2 - C_1 + A_3 + C_2$, ce qui est moins bien si $A_2 + C_1 > A_3 + C_2$.

6.5 Détails du PL primal (Machine à 3 états)

Détaillons à présent le chemin de pensées pour arriver à ce programme linéaire. Tout d'abord, la fonction objectif est facile à comprendre puisqu'il s'agit simplement de minimiser la somme des dépenses au cours du temps (en fonction de l'état dans lequel se trouve la machine) et du coût pour revenir dans l'état ON afin d'exécuter la nouvelle tâche.
Attardons nous désormais sur les contraintes d'intégrité. Pour commencer, la contrainte sur l'état initial signale simplement qu'au début, la machine est dans l'état ON puisqu'elle vient tout juste de finir l'exécution de la dernière tâche. Ensuite, la contrainte $ON_i + VEI_i + OFF_i = 1$ exprime le fait que la machine est dans un unique état à chaque instant (on rappelle que le domaine de définition des variables est {0,1} et les variables sont donc entières). On note qu'il n'est pas utile d'exprimer cette contrainte pour i valant 0 puisqu'elle est déjà exprimé par les contraintes de l'état initial.
Les deux contraintes suivantes vont de pair (pour chaque unité de temps i, $x_{OFF} \geq OFF_i$ et $x_{VEI} \geq VEI_i - x_{OFF}$) et expriment le fait que x représente bien le dernier état, c'est-à-dire que x_{OFF} ne vaut 1 que si le dernier état avant l'arrivée de la tâche est OFF et, de la même manière, x_{VEI} ne vaut 1 que si le dernier état avant l'arrivée de la tâche est VEI. En effet, étant donné que l'on ne peut pas passer d'un état "inférieur" vers un état "supérieur" avant l'arrivée de la tâche, alors si la machine est dans l'état OFF à un certain temps i, elle restera dans cet état jusqu'à ce que la tâche arrive. Ainsi, x_{OFF} ne vaut 1 que si OFF_i vaut 1 pour un certain i (cela aura pour conséquence que la machine reste dans l'état OFF pour tous les pas de temps suivants), d'où la première contrainte qui indique que si pour un certain i, OFF_i vaut 1 alors $x_{OFF} \geq 1$, c'est-à-dire $x_{OFF} = 1$ et dans le cas contraire, si OFF_i vaut toujours 0 alors la contrainte se réécrit simplement $x_{OFF} \geq 0$, ce qui est toujours vrai et on donnera donc 0 à la variable x_{OFF} puisqu'on est en minimisation.
De la même manière, si la machine est dans l'état VEI à un certain temps i, elle ne pourra que rester dans cet état ou passer dans l'état OFF mais ne pourra pas revenir dans l'état ON avant l'arrivée de la tâche. Ainsi, x_{VEI} ne vaut 1 que si VEI_i vaut 1 pour un certain i et que x_{OFF} vaut 0 (cela signifie en effet que la machine

n'est pas dans l'état OFF à la fin et qu'elle n'a donc jamais été dans l'état OFF auparavant, sinon elle y serait restée, et puisqu'il existe un pas de temps i tel que VEI_i vaut 1, alors on en déduit que la machine est restée dans l'état VEI jusqu'à l'arrivée de la tâche), d'où la seconde contrainte qui indique que si pour un certain i, VEI_i vaut 1 et que x_{OFF} vaut 0, alors $x_{VEI} \geq VEI_i - x_{OFF} = 1$, c'est-à-dire $x_{VEI} = 1$ et dans le cas contraire, si VEI_i vaut 0 et/ou si x_{OFF} vaut 1 alors la contrainte se réécrit simplement $x_{VEI} \geq 0$ ou $x_{VEI} \geq -1$, ce qui est toujours vrai et on donnera donc 0 à la variable x_{VEI} puisqu'on est en minimisation.
Notons que nous aurions pu ré-écrire ces contraintes sur le x comme la contrainte suivante : $x_{OFF} + x_{VEI} + ON_N = 1$, où N est le dernier pas de temps avant l'arrivée de la tâche. Cette contrainte traduit bien la même contrainte sur les x, c'est-à-dire que le x correspond au dernier état avant l'arrivée de la tâche, mais elle aurait posé problème dans le cas en ligne dans lequel on ne connaît pas la valeur de N.
On note qu'il n'est pas utile d'exprimer ces deux contraintes pour i valant 0 puisque, selon les contraintes de l'état initial, elles reviendront simplement à dire que $x_{OFF} \geq 0$ et $x_{OFF} + x_{VEI} \geq 0$, ce qui est déjà exprimé par le domaine de définition des variables.
Enfin, les deux dernières contraintes vont également de pair (pour chaque unité de temps i et pour tout j > i, $ON_i \geq ON_j$, et $OFF_i \leq OFF_j$) et expriment le fait que l'on ne peut pas passer d'un état "inférieur" vers un état "supérieur" avant l'arrivée de la tâche. En effet, en fixant pour chaque unité de temps i et pour tout j > i, $ON_i \geq ON_j$, alors si ON_i vaut 0, la contrainte se ré-écrit $0 \geq ON_j$, c'est-à-dire $ON_j = 0$ et cela traduit bien le fait qu'à partir du moment où la machine n'est plus dans l'état ON, elle ne le sera plus jamais avant l'arrivée de la tâche. Autrement, si ON_i vaut 1, la contrainte se ré-écrit $1 \geq ON_j$, ce qui est toujours vrai et on pourra donc choisir de rester dans l'état ON ou non. Concernant la dernière contrainte, si l'on fixe pour chaque unité de temps i et pour tout j > i, $OFF_i \leq OFF_j$, alors si OFF_i vaut 1, la contrainte se ré-écrit $1 \leq OFF_j$, c'est-à-dire $OFF_j = 1$ et cela traduit bien le fait qu'à partir du moment où la machine est dans l'état OFF, elle le restera jusqu'à l'arrivée de la tâche. Autrement, si OFF_i vaut 0, la contrainte se ré-écrit $0 \leq OFF_j$, ce qui est toujours vrai et on pourra donc choisir de passer dans l'état OFF ou non. On a donc traduit qu'à partir du moment où l'on n'est plus dans l'état ON, on n'y revient plus et qu'à partir du moment où l'on est dans l'état OFF, on y reste. On n'a donc pas besoin d'exprimer de contrainte pour l'état VEI puisque l'on déduit des deux contraintes précédentes que si l'on n'est ni dans l'état ON ni dans l'état OFF, alors on est dans l'état VEI et que cela ne peut pas arriver avant d'être dans un état ON (puisque pour atteindre l'état VEI on a du quitter l'état ON et qu'on ne peut pas y revenir) ni après avoir été dans un état OFF (sinon cela voudrait dire qu'il y a eu un état OFF par le passé qui n'a pas été suivi par des états OFF exclusivement et cela serait absurde comme on l'a vu). On note ici qu'il est important que l'indice i de ces deux contraintes commence à zéro afin de créer la continuité entre l'état initial et les états futurs. Notons également que l'indice i s'arrête à N-1 puisque sinon aucun j > i n'existe.
Précisons désormais une piste qui n'a pas abouti pour formuler cette contrainte de "décroissance" des états. Rappelons que le but était de formaliser les contraintes $\begin{cases} \text{Si } OFF_i = 1, \text{ alors } \forall j \geq i, ON_j = 0 \text{ et } VEI_j = 0 \\ \text{Si } VEI_i = 1, \text{ alors } \forall j \geq i, ON_j = 0 \end{cases}$ sous forme de contraintes d'intégrité linéaire. La piste était donc d'introduire de nouvelles variables T_i représentant le coût de transition de l'état de la machine au temps i à l'état de la machine au temps $i+1$. Ainsi, on aurait $T_i = \alpha_1 . \left\lfloor \frac{ON_i + VEI_{i+1}}{2} \right\rfloor + \alpha_2 . \left\lfloor \frac{VEI_i + OFF_{i+1}}{2} \right\rfloor + \alpha_3 . \left\lfloor \frac{ON_i + OFF_{i+1}}{2} \right\rfloor + \alpha_4 . \left\lfloor \frac{VEI_i + ON_{i+1}}{2} \right\rfloor + \alpha_5 . \left\lfloor \frac{OFF_i + VEI_{i+1}}{2} \right\rfloor + \alpha_6 . \left\lfloor \frac{OFF_i + ON_{i+1}}{2} \right\rfloor$ où α_4, α_5, et α_6 seront fixés arbitrairement grand (puisqu'on ne veut pas pouvoir changer d'état vers un état supérieur). Afin d'être cohérent avec l'énoncé, on peut fixer $\alpha_4 = C_1$, $\alpha_5 = C_2 - C_1$, et $\alpha_6 = C_2$, ce qui empêche de la même manière un changement d'état vers un état supérieur puisqu'il ne sert à rien de payer le coût vers un état supérieur avant l'arrivée de la tâche.
En effet, cette formulation convient puisque chaque $\left\lfloor \frac{E_i + F_{i+1}}{2} \right\rfloor$ où E_i et F_{i+1} sont les états ON, OFF, ou VEI (cela revient exactement au même dans ce raisonnement) ne vaut 1 que si à la fois E_i et F_{i+1} valent 1, c'est à dire si on a bien fait une transition de l'état E au temps i vers l'état F au temps $i+1$. Il y a donc, pour chaque pas de temps i, $T_i = \alpha_j$ où le j est déterminé en fonction du changement d'état effectué. Il suffirait ensuite d'ajouter $\sum_{i=1}^{N-1} T_i$ à la fonction objectif du programme linéaire afin de forcer le programme à éviter d'utiliser des transitions trop coûteuses (puisqu'on est en minimisation) et donc rendre impossible les transitions vers des états supérieurs. Le problème que l'on voit pour cette piste est l'utilisation de la partie entière inférieure, qui n'est pas linéaire. On ne peut donc pas utiliser cette formulation telle qu'elle et c'est pour cette raison que nous avons opté pour la solution vue plus haut.

6.6 Simulations avec l'approche "Primal-Dual" (Machine à 3 états)

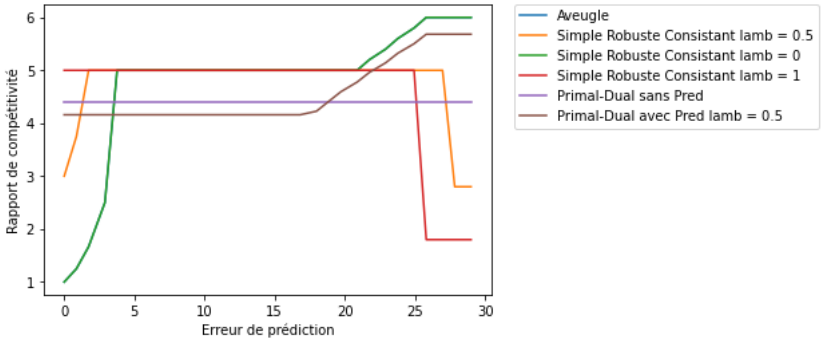

FIGURE 9 – Comparaison des algorithmes en fonction de l'erreur de prédiction (avec "Primal-Dual")

Nous affichons ci-dessus les simulations pour les algorithmes "Primal-Dual" (avec comme valeurs arbitraires $FAIBLE = 2$ et $ELEVE = 100$), bien que, comme on l'a vu plus haut, ces algorithmes n'apportent pas vraiment de solution à notre problème puisqu'il met en place des simplifications telles que le caractère flottant des variables (et par conséquent le fait que la machine puisse être dans plusieurs états à la fois). De plus, le ratio de compétitivité des algorithmes "Primal-Dual" n'est pas très bon ce qui peut être expliqué par la non convergence de la formule de mise à jour. Néanmoins, il vaut sûrement le coup de chercher un algorithme plus proche des formulations des programmes linéaires étant donné les résultats obtenus pour le problème de la rentabilité des skis.

6.7 Détails du PL primal (Machine à k états)

Détaillons à présent le chemin de pensées pour arriver à ce programme linéaire. Tout d'abord, la fonction objectif est facile à comprendre puisqu'il s'agit simplement de minimiser la somme des dépenses au cours du temps (en fonction de l'état dans lequel se trouve la machine) et du coût pour revenir dans l'état 1 afin d'exécuter la nouvelle tâche.
Attardons nous désormais sur les contraintes d'intégrité. Pour commencer, la contrainte sur l'état initial signale simplement qu'au début, la machine est dans l'état 1 puisqu'elle vient tout juste de finir l'exécution de la dernière tâche. La contrainte $\sum_{i=2}^{k} et_{(i,0)} = 0$ signifie simplement que la machine n'est dans aucun état différent de l'état 1 au temps 0. En effet, puisque chaque variable $et_{(i,j)}$ est positive, si leur somme vaut 0 alors c'est que toutes ces variables sont nulles. Ensuite, la contrainte $\sum_{i=1}^{k} et_{(i,j)} = 1$ exprime le fait que la machine est dans un unique état à chaque instant (on rappelle que le domaine de définition des variables est {0,1} et les variables sont donc entières). On note qu'il n'est pas utile d'exprimer cette contrainte pour j valant 0 puisqu'elle est déjà exprimé par les contraintes de l'état initial.
La contrainte suivante (pour chaque unité de temps j (de 1 à N) et pour tout i (de 1 à k), $x_i \geq et_{(i,j)} - \sum_{l=i+1}^{k} x_l$) exprime le fait que x représente bien le dernier état, c'est-à-dire que x_i ne vaut 1 que si le dernier état avant l'arrivée de la tâche est i. En effet, étant donné que l'on ne peut pas passer d'un état "inférieur" vers un état "supérieur" avant l'arrivée de la tâche, alors si la machine est dans l'état i à un certain temps j, elle ne pourra que rester dans cet état ou passer dans un état inférieur mais ne pourra pas revenir dans un état supérieur avant l'arrivée de la tâche. Ainsi, x_i ne vaut 1 que si $et_{(i,j)}$ vaut 1 pour un certain j et que x_l vaut 0 pour tout $l > i$ (cela signifie en effet que la machine n'est dans aucun état $l > i$ à la fin et qu'elle n'a donc jamais été dans un état inférieur à i auparavant, sinon elle serait toujours dans un état inférieur à i (puisqu'elle ne peut pas aller vers un état supérieur), et puisqu'il existe un pas de temps j tel que $et_{(i,j)}$ vaut 1, alors on en déduit que la machine est restée dans l'état i jusqu'à l'arrivée de la tâche), d'où la contrainte qui indique que si pour un certain j, $et_{(i,j)}$ vaut 1 et que x_l vaut 0 pour tout $l > i$, alors $x_i \geq et_{(i,j)} - \sum_{l=i+1}^{k} x_l = 1$, c'est-à-dire $x_i = 1$ et dans le cas contraire, si $et_{(i,j)}$ vaut 0 et/ou si un des x_l vaut 1 alors la contrainte se réécrit simplement $x_i \geq 0$ ou $x_i \geq -1$, ce qui est toujours vrai et on donnera donc 0 à la variable x_i puisqu'on est en minimisation.
Notons que nous aurions pu ré-écrire cette contrainte sur le x comme les contraintes suivantes : pour tout i (de 1 à k), $x_i = et_{(i,N)}$, où N est le dernier pas de temps avant l'arrivée de la tâche. Cette contrainte traduit bien la même contrainte sur les x, c'est-à-dire que le x correspond au dernier état avant l'arrivée de la tâche, mais elle aurait posé problème dans le cas en ligne dans lequel on ne connaît pas la valeur de N.
On note qu'il n'est pas utile d'exprimer cette contrainte pour j valant 0 puisque, selon les contraintes de l'état initial, elle reviendra simplement à dire que pour tout i (de 1 à k), $x_i \geq et_{(i,0)} - \sum_{l=i+1}^{k} x_l$ soit pour $i = 1$, $x_1 \geq 1 - \sum_{l=i+1}^{k} x_l$, ce qui n'apporte pas d'information puisqu'on considère qu'au moins un pas de temps sépare deux tâches et pour tout $i > 1$, $x_i \geq -\sum_{l=i+1}^{k} x_l$, ce qui est déjà exprimé par le domaine de définition des

variables puisque $x_i \geq 0 \ \forall i$.
Enfin, la dernière contrainte (pour tout i (de 1 à k), pour chaque unité de temps j (de 1 à N-1), et pour tout l > j (de j+1 à N), $et_{(i,j)} \geq \sum_{r=1}^{i-1} et_{(r,l)} - \sum_{h=1}^{i-1} et_{(h,j)})$ exprime le fait que l'on ne peut pas passer d'un état "inférieur" vers un état "supérieur" avant l'arrivée de la tâche. On veut exprimer le fait que si, pour un état i (de 1 à k), pour un pas de temps j (de 0 à N), $et_{(i,j)} = 1$, alors $\forall l > j, \sum_{r=1}^{i-1} et_{(r,l)} = 0$, c'est-à-dire que la machine ne pourra pas être dans un état $r < i$ dans un instant futur puisqu'elle est déjà dans l'état i au moment présent et qu'elle ne peut pas aller vers un état supérieur. En effet, en fixant pour chaque état i, pour chaque unité de temps j et pour tout l > j, $et_{(i,j)} \geq \sum_{r=1}^{i-1} et_{(r,l)} - \sum_{h=1}^{i-1} et_{(h,j)}$, alors si $et_{(i,j)}$ vaut 0, la contrainte se ré-écrit $0 \geq \sum_{r=1}^{i-1} et_{(r,l)} - \sum_{h=1}^{i-1} et_{(h,j)}$, c'est-à-dire si $\sum_{h=1}^{i-1} et_{(h,j)} = 0$ (dans ce cas la machine n'est ni dans l'état i, ni dans un état $h < i$), la contrainte se réécrit $0 \geq \sum_{r=1}^{i-1} et_{(r,l)}$, soit $\sum_{r=1}^{i-1} et_{(r,l)} = 0$, soit toutes les variables $et_{(r,l)}$ pour r allant de 1 à $i-1$ valent 0 et cela traduit bien le fait qu'à partir du moment où la machine n'est plus dans l'état i ni dans un état $h < i$ (mais dans un état inférieur), elle ne le sera plus jamais avant l'arrivée de la tâche. Maintenant, toujours dans le cas où $et_{(i,j)}$ vaut 0, si $\sum_{h=1}^{i-1} et_{(h,j)} = 1$ (dans ce cas la machine est dans un état $h < i$), la contrainte se réécrit $1 \geq \sum_{r=1}^{i-1} et_{(r,l)}$, ce qui est toujours vrai puisque la machine est dans un unique état à la fois (cette somme vaudra donc 1 si la machine est dans un état entre 1 et $i-1$ au temps l et vaudra 0 sinon).
Autrement, si $et_{(i,j)}$ vaut 1, la contrainte se ré-écrit $1 \geq \sum_{r=1}^{i-1} et_{(r,l)} - \sum_{h=1}^{i-1} et_{(h,j)}$, soit $1 \geq \sum_{r=1}^{i-1} et_{(r,l)}$ puisque la machine ne peut être que dans un état à la fois (à l'instant j la machine est dans l'état i puisque $et_{(i,j)}$ vaut 1 et n'est donc dans aucun autre état, soit $\forall h < i, et_{(h,j)} = 0$), ce qui est toujours vrai puisque la machine est dans un unique état à la fois (cette somme vaudra donc 1 si la machine est dans un état entre 1 et $i-1$ au temps l et vaudra 0 sinon). On note ici qu'il est important que l'indice j de cette contrainte commence à zéro afin de créer la continuité entre l'état initial et les états futurs. Notons également que l'indice j s'arrête à N-1 puisque sinon aucun $l > j$ n'existe. Notons finalement que l'indice i peut aussi bien commencer à 2 puisque pour $i = 1$, la contrainte devient : pour chaque unité de temps j (de 1 à N-1), et pour tout l > j (de j+1 à N), $et_{(1,j)} \geq \sum_{r=1}^{0} et_{(r,l)} - \sum_{h=1}^{0} et_{(h,j)} = 0$ et est déjà exprimée par le domaine de définition de $et_{(i,j)}$.

6.8 Simulations avec l'approche "Primal-Dual" (Machine à k états)

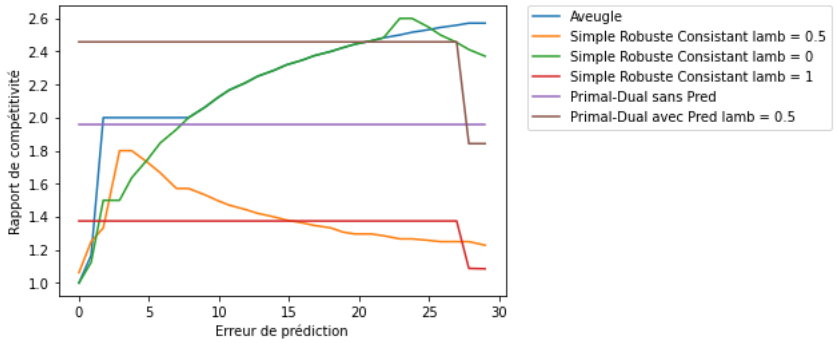

FIGURE 10 – Comparaison des algorithmes en fonction de l'erreur de prédiction pour k=3

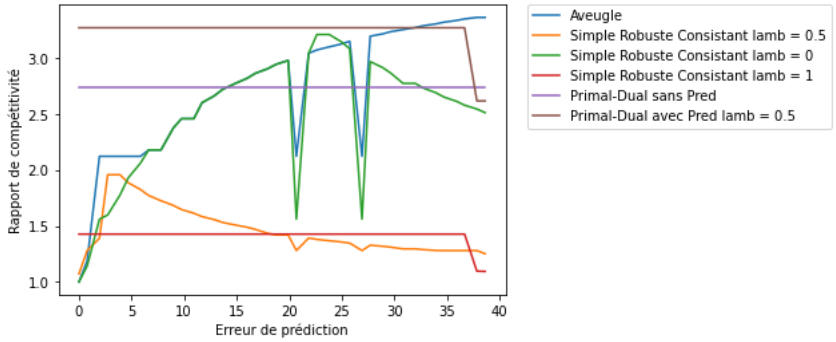

FIGURE 11 – Comparaison des algorithmes en fonction de l'erreur de prédiction pour k=4

Nous affichons ci-contre les simulations pour les algorithmes "Primal-Dual" (avec comme valeurs arbitraires $FAIBLE = 2$ et $ELEVE = 100$), bien que, comme on l'a vu plus haut, ces algorithmes n'apportent pas

vraiment de solution à notre problème puisqu'il met en place des simplifications telles que le caractère flottant des variables (et par conséquent le fait que la machine puisse être dans plusieurs états à la fois). De plus, le ratio de compétitivité des algorithmes "Primal-Dual" n'est pas très bon ce qui peut être expliqué par la non convergence de la formule de mise à jour. Néanmoins, il vaut sûrement le coup de chercher un algorithme plus proche des formulations des programmes linéaires étant donné les résultats obtenus pour le problème de la rentabilité des skis.

6.9 Planning

Semaine 1 :
Lecture des documents 1 à 4, 6 et rédaction de l'introduction.

Semaine 2 :
Lecture des documents 5, 7 et 8, rédaction de la partie sur le problème de la rentabilité des skis (et réflexion pour la partie démonstration) et la partie idée.

Semaine 3 :
Réflexion sur la mise en forme du PL pour le problème d'économie d'énergie pour une machine à états, rédaction de cette partie.

Semaine 4 :
Rédaction de la partie sur les algorithmes simples pour le problème de la rentabilité des skis et réflexion sur comment adapter ces algorithmes pour le problème d'économie d'énergie pour une machine à états, puis rédaction de ces parties.

Semaine 5 :
Encodage des algorithmes pour le problème de la rentabilité des skis, ajustement du programme linéaire primal pour le problème d'économie d'énergie pour une machine à états et réflexion sur la formulation du dual de ce programme. Modification de certains détails dans les algorithmes simples pour le problème d'économie d'énergie pour une machine à états. Réflexion sur le ratio de compétitivité de l'algorithme robuste et consistant avec prédiction pour le problème d'économie d'énergie pour une machine à 2 états et rédaction de la preuve.

Semaine 6 :
Rédaction de la partie sur le dual du PL concernant le problème d'économie d'énergie pour une machine à 3 états. Mise en place de simulations afin de comparer les algorithmes pour le problème de la rentabilité des skis. Encodage des algorithmes simples pour le problème d'économie d'énergie pour une machine à 3 états et mise en place de simulations les comparant. Rédaction des parties simulations.

Semaine 7 :
Réflexion sur l'extension du problème d'économie d'énergie pour une machine à k états et rédaction de cette partie : présentation du problème, réflexion sur la mise en forme du PL, réflexion sur comment adapter les algorithmes simples à ce problème, réflexion sur la formulation du dual de ce problème. Modification de certains détails erronés sur la formulation du PL pour le problème d'économie d'énergie pour une machine à 3 états.

Semaine 8 :
Rédaction de la partie sur le dual et réglage de problèmes d'indexage sur l'expression de ce PL. Encodage des algorithmes simples pour le problème d'économie d'énergie pour une machine à k états et mise en place de simulations les comparant. Rédaction de la partie simulation. Réglage de problèmes sur les simulations (pour 3 et k états).

Semaine 9 :
Réflexion sur la preuve de robustesse pour l'algorithme simple du problème d'économie d'énergie pour une machine à 3 états. Réflexion sur la mise en place des algorithmes Primal-Dual pour le problème d'économie d'énergie pour une machine à 3 et k états. Mise en forme du rapport final.

6.10 Autre idée concernant les Learning Augmented Algorithms

Considérons un autre problème dans lequel on a m machines disponibles et n tâches à exécuter. Notre but est de minimiser la date de fin de la dernière tâche exécutée. Pour ce faire, nous avons à notre disposition un algorithme de liste $(2 - \frac{1}{m})$-approché ne nécessitant pas de connaître la durée des tâches. Ce dernier prend une liste qui détermine un ordre quelconque des tâches en entrée, et à chaque instant où une machine se libère, l'algorithme lui affecte la prochaine tâche de la liste. Un cas particulier de cet algorithme, appelé LPT (*longest processing time first*) est de prendre en entrée non pas une liste quelconque mais une liste dans laquelle les tâches sont triées de la plus longue à la plus courte, ce qui suppose donc de connaître les informations sur la durée des tâches. Cet algorithme est, quant à lui, $(\frac{4}{3} - \frac{1}{3m})$-approché. La preuve de ces affirmations se trouve dans l'article [6].

On peut donc imaginer combiner les deux algorithmes précédents dans le cas où l'on aurait à notre disposition non pas les durées exactes des tâches mais les durées prédites. Cette combinaison donnerait de bons résultats si la prédiction est bonne (grâce au deuxième algorithme combiné) et une performance proche d'un algorithme $(2 - \frac{1}{m})$-approché lorsque les prédictions sont mauvaises (grâce au premier algorithme combiné).

Une autre idée serait de combiner LPPT (*longest predicted processing time first*) et SPPT (*shortest predicted processing time first*) qui sont des algorithmes de listes prenant en entrée respectivement la liste des tâches de longueur prédites de la plus grande à la plus petite et la liste des tâches de longueur prédites de la plus petite à la plus grande. Cette combinaison donnera ainsi de bons résultats si les prédictions sont exactes (grâce à LPPT) et également si les prédictions sont totalement inversées (grâce à SPPT), c'est à dire si la durée la plus longue est prédite comme la plus courte et inversement, et ainsi de suite. Il reste néanmoins une zone d'ombre dans le cas où certaines durées sont bien prédites tandis que d'autres ne le sont pas. En effet, il faut trouver une solution pour ce cas de figure si on ne veut pas avoir de très mauvaises performance. Pour combiner ces deux algorithmes on peut penser à appliquer l'un ou l'autre avec une probabilité de α et $(1 - \alpha)$ respectivement.