Contrôle du raisonnement progressif pour des plans d'actions non linéaires

Stéphane Cardon et Abdel-Illah Mouaddib
CRIL-Faculté de Lens-Université d'Artois
Rue de l'université, S. P. 18
62307 Lens Cedex France
{cardon, mouaddib}@cril.univ-artois.fr

Résumé *Le raisonnement progressif est une technique de raisonnement qui permet à un système de construire une solution d'un problème en structurant la solution en une hiérarchie de niveaux. Cette technique offre un compromis entre la consommation des ressources et le résultat obtenu.*

Cet article se focalise sur le contrôle du raisonnement progressif dans des plans d'actions non linéaires. Plus précisément, nous nous focalisons sur le contrôle d'un robot explorateur qui doit évoluer dans un environnement dynamique avec des ressources limitées. Nous montrerons comment construire un contrôle optimal pour un plan d'action non linéaire en résolvant le problème décisionnel de Markov correspondant. Finalement nous montrerons l'efficacité de notre méthode par des expérimentations et nous énoncerons un théorème de réduction du temps de calcul de ce contrôle.
mot-clés : *Planification sous incertitude, raisonnement avec des ressources limitées, contrôle*

1 Introduction

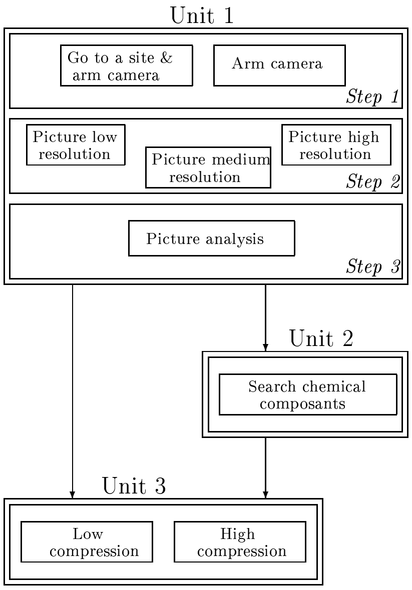
Les robots d'exploration doivent fonctionner en respectant de fortes contraintes sur les ressources telles que le temps de fonctionnement, l'énergie, la capacité de stockage et le temps de communication des résultats. Dans un souci d'optimisation, un robot possède une ou plusieurs méthodes pour accomplir chaque niveau du plan d'action. Les différentes alternatives offrent un compromis entre la consommation des ressources et la qualité du résultat.

Dans ce contexte, le raisonnement progressif est une technique adéquate. En effet, il structure la solution de chaque problème vue comme une hiérarchie de tâches. Ces dernières contribuent à la qualité générale de la solution. Cette technique partage les mêmes motivations que les techniques de raisonnement à ressources limitées telles que *"flexible computation"*[1], *"anytime algorithms"*[2], *"imprecise computation"*[3, 4] et *"design-to-time"*[5]. Ces techniques offrent un compromis entre la qualité de la solution et les ressources consommées.

Dans des travaux précédents, Abdel-Illah Mouaddib et Shlomo Zilberstein, [6], [7] ont présenté une approche du contrôle du raisonnement progressif. Ce problème est de décider combien de ressources peuvent être allouées à chaque unité de raisonnement progressif. Elle se base sur la formulation de ce problème en un problème de contrôle d'un processus décisionnel de Markov (PDM). Ils ont d'ailleurs implanté un algorithme efficace [7].

Cependant, cet algorithme (et la technique sur laquelle il se base) ne travaille que sur des suites de problèmes indépendants, où chaque problème n'est qu'une succession de tâches à exécuter ou non.

FIG. 1: Plan d'action d'un robot explorateur



Pour nos robots, les problèmes ne sont pas forcément linéaires (cf. figure 1). Un des problèmes à résoudre serait d'aller sur un site, d'y prendre une photo, de rechercher l'existence de pierres et dans ce cas, faire des recherches des composantss chimiques de la pierre, et enfin, compresser les données scientifiques trouvées. De plus, il peut prendre des photos basse, moyenne ou haute résolution et compresser fortement ou non les données. Cet exemple présente de nouveaux aspects non traités jusqu'alors :

– **Non linéarité du raisonnement progressif :** Les unités à traiter ne sont plus linéaires mais des graphes dirigés et acycliques de tâches. Seul un chemin sera réellement traité.

- **Plan sous forme de graphe dirigé et acyclique :** Le plan est lui aussi un graphe dirigé et acyclique d'unités de raisonnement progressif. De nouveau, seul un chemin sera traité.
- **Inter-dépendance des tâches :** Le traitement des tâches dépend du résultat obtenus lors du traitement des tâches précédentes. Par exemple, faire des recherches de composant chimiques sur des pierres d'un site dépend des résultats de l'analyse de l'image.

Donc, nous avons étendu cette méthode à des graphes de problèmes dépendant de la solution de leur prédécesseur et, à des problèmes dont l'unité de raisonnement progresif n'est pas linéaire mais représentant aussi un graphe de possibilités.

La section 2 définit formellement le problème. Dans la section 3, nous développerons un contrôleur optimal pour ce type de plan d'action. La section 4 nous montrera l'efficacité de notre méthode par des expérimentations. Enfin, nous discuterons des résultats et nous parlerons des perspectives de ce travail.

2 Problème du contrôle du raisonnement progressif

Dans la section précédente, nous faisons référence aux notions de raisonnement progressif, d'unités de raisonnement progressif et de plan d'action. Avant de définir ces notions d'une manière formelle, nous en donnerons une définition intuitive.

2.1 Définitions intuitives

Le raisonnement progressif est une technique de raisonnement qui construit de façon incrémentale la solution d'un problème en utilisant une hiérarchie de niveaux où chaque niveau contribue à la solution. Nous pouvons voir un niveau comme une étape de raisonnement et la hiérarchie de niveaux comme le schéma du raisonnement nous amenant à la solution. Nous noterons URP (unité de raisonnement progressif) cette hiérarchie de niveaux. Remarquons qu'à chaque niveau, nous pouvons associer une utilité d'exécution qui est une mesure abstraite de la qualité donnée par un niveau dans l'élaboration de la solution.

Avec cette définition intuitive, le plan d'action d'un robot peut être vu comme un ensemble ordonné d'unités de raisonnement progressif. Plus généralement, le plan d'action peut être non linéaire dans le sens où il s'agit d'un graphe orienté sans cycles d'unités de raisonnement progressif. Une URP peut avoir plusieurs successeurs qui dépendent de sa solution.

Pour notre robot explorateur, nous avons trois URPs qui correspondent à la prise d'une photo dans un site et son analyse, la recherche des composants chimiques d'une pierre et la compression des données recueillies. La deuxième URP dépend du résultat de l'analyse de la première. Nous pourrions transformer ce plan en une suite linéaire d'URPs mais dans ce cas nous perdrions de la généralité et cette notion de dépendance entre les unités. En fait, la non linéarité du plan d'action et la notion de dépendance permettent de traiter un plus grand nombre de problèmes.

Le problème du robot explorateur (ou problème du contrôle du raisonnement progressif) est de choisir l'ensemble des tâches qu'il devra accomplir en maximisant l'utilité globale et en respectant les contraintes sur ses ressources.

2.2 Définitions formelles

Formalisons ces définitions en nous focalisant sur un type de ressource : le temps de fonctionnement du robot.

Définition 1 *Un **plan d'action** $\mathcal{P}$ est un ensemble ordonné d'unités de raisonnement progressif, $\mathcal{P} = \{P_1, P_2, ..., P_{\mathcal{N}}\}$.*

Définition 2 *Une **unité de raisonnement progressif** (URP), notée P_i, se compose d'un ensemble de niveaux, $n_i = \{N_{i,1}, N_{i,2}, ..., N_{i,\#n_i}\}$, d'une "deadline" d_i et d'un ensemble de successeurs $s_i = \{succ(j_1), ..., succ(j_{\#s_i})\}$, où $succ(j_l) = ([MinQ_{j_l} - MaxQ_{j_l}], P_{j_l})$ et $[MinQ_{j_l} - MaxQ_{j_l}]$ correspond aux utilités minimale et maximale requises pour passer à l'unité suivante P_{j_l}.*

Définition 3 *Un **niveau** j d'une unité i, notée $N_{i,j}$, se compose d'un ensemble de tâches, $t_{i,j} = \{T_{i,j,1}, T_{i,j,2}, ..., T_{i,j,\#t_{i,j}}\}$ et d'un attribut "skippable". Ce dernier indique s'il est possible ou non de sauter l'exécution d'une tâche quelconque de ce niveau.*

Définition 4 *Une **tâche** k d'un niveau j d'une unité i, notée $T_{i,j,k}$, correspond à un module à exécuter, elle possède une utilité (ou qualité) $q_{i,j,k}$ et une distribution probabiliste des ressources consommées par son exécution, $d_{i,j,k} = \{(rc_{i,j,k}, pr_{i,j,k})_1, ..., (rc_{i,j,k}, pr_{i,j,k})_{\#d_{i,j,k}}\}$.*
Nous noterons $T_{i,0,1}$ la tâche spéciale (jamais exécutée) servant à marquer le début d'une nouvelle unité.

Définition 5 *Le problème du **contrôle du raisonnement progressif** est de sélectionner un ensemble de tâches afin de maximiser l'utilité globale.*

2.3 Selection de l'ensemble des tâches

Avec ces definitions, mettons-nous à la place du robot. Supposons que nous sommes dans un niveau quelconque $N_{i,j}$ et que nous venons d'exécuter la tâche $T_{i,j,k}$. Nous avons maintenant trois choix :

- soit nous exécutons une tâche $T_{i,j+1,k'}$ du niveau suivant $N_{i,j+1}$, s'il existe ;
- soit nous décidons de ne pas exécuter le niveau suivant $N_{i,j+1}$, s'il existe et s'il est "skippable" (en un mot, nous voulons passer au niveau suivant sans exécuter une tâche en particulier) ;
- soit nous décidons d'arrêter notre travail pour cette unité P_i et nous passons à l'unité suivante $P_{succ(i)}$.

Ces choix contribuent à la construction de l'ensemble des tâches et l'utilité globale peut-être donnée par la somme des qualités associées à chaque tâche exécutée.

Dans ce cadre, nous pouvons voir le problème du contrôle du raisonnement progressif comme un graphe de tâches exécutées ou non où chaque transition correspond aux choix définis précédemment. En fait, *le choix de l'action suivante* (lorsque nous nous trouvons dans un état en particulier) *ne dépend que de cet état*, ce qui vérifie la **propriété markovienne**. Donc, le problème du contrôle du raisonnement progressif peut être vu comme le problème de contrôle d'un processus décisionnel de Markov (PDM).

2.4 Processus décisionnel de Markov

Le processus décisionnel de Markov correspondant à notre problème se définit comme suit.

2.4.1 Etats

D'un point de vue intuitif, un état du processus décisionnel de Markov correspond à la tâche courante dans le plan d'action et aux ressources restantes du système.

Définition 6 *Un **état**, noté $[T_{i,j,k}; RR]$ correspond à la dernière tâche $T_{i,j,k}$ exécutée ou non et aux ressources restantes du système.*

Définition 7 *La **qualité accumulée d'un état** est la somme des qualités associées aux tâches exécutées pour tous les état précédents.*

Définition 8 *Un **état accessible** est un état de la forme $[T_{i',0,k'}; RR']$ dont le prédécesseur $[T_{i,j,k}; RR]$ possède une qualité accumulée appartenant à $[MinQ_l - MaxQ_l]$ où $([MinQ_l - MaxQ_l], P_l) \in succ_i$ et $P_l = P_{k'}$.*

Définition 9 *Un **état final** est un état dont les ressources restantes sont nulles ou ne possèdant plus d'états successeurs accessibles (c'est à dire, l'unité courante n'admet plus de successeurs à cause de la qualité accumulée de l'état courant).*

2.4.2 Transitions

D'un point de vue intuitif, une transition correspond à l'action qui permet de passer d'un état e_1 à un état e_2. Donc, nous pouvons avoir trois types de transitions, E, S et D.

Définition 10 *Une **transition** $\mathbf{E_{i,j,k}}$ de type E correspond à l'exécution de la prochaine tâche $T_{i,j,k}$. Nous la noterons $[T_{i,j-1,k'}; RR] \xrightarrow{E_{i,j,k}} [T_{i,j,k}; RR - \Delta R]$, où ΔR représente les ressources consommées par l'exécution de $T_{i,j,k}$.*

Définition 11 *Une **transition** $\mathbf{S_{i,j}}$ de type S correspond au saut de la prochaine tâche. Nous la noterons $[T_{i,j-1,k'}; RR] \xrightarrow{S_{i,j}} [T_{i,j,k}; RR]$.*

Définition 12 *Une **transition** $\mathbf{D_i}$ de type D correspond au passage à une autre unité. Nous la noterons $[T_{i',j,k}; RR] \xrightarrow{D_i} [T_{i,0,1}; RR]$., où P_i est un successeur accessible de $P_{i'}$.*

2.4.3 Règles de transitions du PDM

Maintenant que nous avons défini les états et transitions du PDM, voyons les règles de transitions du processus.

La formule *Pr(état f/état e, transition t)=probabilité p* se lit : la probabilité d'arriver en f en partant de e par la transition t est p.

Plaçons-nous dans l'état $e = [T_{i,j-1,k'}; RR]$.

Règle 1 *Transitions $E_{i,j,k}$:*
- *$\Delta R \leq RR$, les ressources consommées par la prochaine tâche sont inférieures ou égales aux ressources restantes du système, $Pr([T_{i,j,k}; RR - \Delta R]|e, E_{i,j,k}) = Pr(rc_{i,j,k} = \Delta R)$.*
- *$\Delta R > RR$, les ressources consommées excédent les ressources restantes du système, $Pr([T_{i',0,1}; d_{i'} - d_i]|e, E_{i,j,k}) = Pr(rc_{i,j,k} = \Delta R)$ où $[T_{i',0,1}; d_{i'} - d_i]$ est accessible par e.*

Règle 2 *Transitions $S_{i,j}$ (si $s_{i,j} = 1$) :*
$Pr([T_{i,j,k}; RR]|e, S_{i,j}) = 1$.

Règle 3 *Transitions D_i :*
$Pr([T_{i',0,1}; d_{i'} - d_i + RR]|e, D_i) = 1$ où $[T_{i',0,1}; d_{i'} - d_i + RR]$ est accessible par e.

2.5 Valeur d'un état

La valeur associée à chaque état $V([T_{i,j,k}; RR])$ est une mesure de l'utilité de cet état dans le PDM. Elle se calcule récursivement en fonction des états successeurs.

Pour cela, prenons l'état $e = [T_{i,j-1,k'}; RR]$ et notons $\mathcal{E}_\mathcal{A} = \{$*états accessibles par e*$\}$ et $ea_{i'} \in \mathcal{E}_\mathcal{A}$ un état accessible par e dont la tâche associée appartient à l'unité $P_{i'}$:
- Transitions $E_{i,j,k}$

$$\begin{aligned}
\alpha_E &= \sum_{\Delta R \leq RR} (Pr(rc_{i,j,k} = \Delta R) \\
&\quad \cdot V([T_{i,j,k}; RR - \Delta R])) \\
&\quad + q_{i,j,k} \\
\alpha_F &= \max_{\forall i', ea_{i'} \in \mathcal{E}_\mathcal{A}} \sum_{\Delta R > RR} (Pr(rc_{i,j,k} = \Delta R) \\
&\quad \cdot V([T_{i',0,1}; d_{i'} - d_i]))
\end{aligned}$$

– Transitions $S_{i,j}$

$$\alpha_S = s_{i,j} \cdot V([T_{i,j,k}; RR])$$

– Transitions D_i

$$\alpha_D = \max_{\forall i', ea_{i'} \in \mathcal{E}_{\mathcal{A}}} V([T_{i',0,1}; d_{i'} - d_i + RR])$$

La valeur d'un état devient donc :

$$V(e) = \max_{\forall E_{i,j,k}, S_{i,j}, D_i} (\alpha_E + \alpha_F, \alpha_S, \alpha_D) \tag{1}$$

3 Contrôleur à base de PDM

Le contrôleur à base de PDM que nous avons implémenté crée récursivement tous les états du PDM et calcule par "backtrack" la valeur de chaque état. Tout en calculant cette valeur, il marque les transitions optimales de chaque état (correspondant aux transitions rendant V optimal).

Théorème 1 (états créés) *Le nombre d'états créés est borné par :*

$$\mathcal{N} \cdot \max_{\forall P_i} d_i \cdot (\max_{\forall P_i} \#n_i)^2 \cdot \max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \tag{2}$$

Preuve:[états créés] En premier lieu, nous nous plaçons dans un plan d'action linéaire qui représente le pire cas que nous pouvons obtenir.

Remarquons que deux états ayant le même nombre de ressources et dont les tâches associées appartiennent au même niveau, donnent les mêmes états successeurs. Donc, pour ne pas rentrer dans une complexité exponentielle (si nous ne faisons pas attention à ce détail), il faut tenir compte des états déjà créés.

Etant donné que les ressources utilisables sont bornées et que le nombre de tâches est borné, nous obtenons un nombre borné d'états. De plus, comme deux états sont différents si leurs tâches associées sont différentes ou leurs ressources restantes sont différentes, nous obtenons $\sum_{\forall P_i} (d_i \cdot \#n_i \cdot \sum_{\forall N_{i,j}} \#t_{i,j})$ états différents. Ce qui nous donne la borne maximale suivante :
$\mathcal{N} \cdot \max_{\forall P_i} d_i \cdot (\max_{\forall P_i} \#n_i)^2 \cdot \max_{\forall P_i, N_{i,j}} \#t_{i,j}$ ■

Théorème 2 (temps création) *Le temps de création du PDM est borné par :*

$$\begin{aligned} \mathcal{N} \quad & \cdot (\max_{\forall P_i} d_i)^2 \cdot \max_{\forall P_i} \#n_i \\ \cdot \quad & [\max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#d_{i,j,k} + 2] \end{aligned} \tag{3}$$

Preuve:[temps création] Le temps de création du PDM est donné par le nombre de transitions créées et le sur-coût engendré par le recherche d'un état (pour ne pas le recréer et ne pas obtenir une complexité exponentielle).

Le nombre d'appels récursifs (pour un état) engendre un nombre équivalent de transitions. Donc, pour un état, le nombre de transitions créées est :

$$\begin{aligned}&[\sum_{k=1}^{\#t_{i,j}} \#d_{i,j+1,k}] + 2\\ &\leq \max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#d_{i,j,k} + 2\end{aligned}$$

Par conséquent, une borne est :

$$\begin{aligned}\mathcal{N} \quad &\cdot \max_{\forall P_i} d_i \cdot \max_{\forall P_i} \#n_i\\ \cdot \quad &[\max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#d_{i,j,k} + 2]\end{aligned}$$

Comme les états sont rangés par unité puis par niveau puis par tâche, la recherche sera de d_i états au maximum. Donc, nous obtenons la borne suivante :

$$\begin{aligned}\mathcal{N} \quad &\cdot (\max_{\forall P_i} d_i)^2 \cdot \max_{\forall P_i} \#n_i\\ \cdot \quad &[\max_{\forall P_i, \forall N_{i,j}} \#t_{i,j} \cdot \max_{\forall P_i, \forall N_{i,j}, \forall T_{i,j,k}} \#d_{i,j,k} + 2]\end{aligned}$$

■

En conclusion, notre contrôleur est en $\mathcal{O}(\mathcal{N} \cdot \max_{\forall P_i} d_i)$ en espace et en $\mathcal{O}(\mathcal{N} \cdot (\max_{\forall P_i} d_i)^2)$ en temps. Enfin, remarquons que si la “deadline” n'est pas en fonction de $\mathcal{N}$, nous obtenons une complexité linéaire.

4 Expérimentations

Nous nous plaçons dans un plan d'action non linéaire comportant $\mathcal{N}$ unités. Chaque unité se décompose entre quatre niveaux. Le premier correspond à se rendre sur site, le deuxième à armer la caméra, le troisième à prendre une photo (basse, moyenne ou haute résolution) et le dernier à compresser (faiblement ou non) les données recueillies.

Nous supposons qu'il faut en moyenne 32 secondes pour se rendre sur le site, armer la caméra, prendre une photo de moyenne résolution et compresser faiblement les données. La “deadline” accordée pour chaque unité est égale à 32 secondes fois la profondeur de cette unité dans le plan d'action. La complexité que nous obtenons, dans ce cas de figure, est en $\mathcal{O}(\mathcal{N} \cdot \log_2 \mathcal{N})$ en espace et en $\mathcal{O}(\mathcal{N} \cdot (\log_2 \mathcal{N})^2)$ en temps.

Lors du calcul de la politique optimale, notre algorithme alloue X ressources si la tâche en consomme X. Prenons maintenant des paquets de u ressources. L'algorithme ne va plus allouer X ressources mais $\lceil \frac{X}{u} \rceil$ paquets. La figure 2 et la table 1 nous montrent l'impact de la consommation de paquets de ressources.

Les résultats obtenus (pour des paquets de taille 2) nous montrent que le temps est divisé par 4 et le nombre d'états créés par 2 (environ). Ces résultats ont été obtenus sur un pentium III 650 avec 192 MO de RAM (Cf. figure 2).

TAB. 1: Erreur engendrée par la consommation de paquets de ressources

$\mathcal{N}$	Valeur (taille=1)	Valeur (taille=2)	Erreur
20	94.38	90.80	3.79%
40	133.76	129.88	2.90%
60	166.76	160.71	3.63%
80	197.37	187.95	4.77%
100	219.37	209.51	4.49%

En fait, nous pouvons généraliser ce résultat :

Théorème 3 ***Si*** *EC est le nombre d'états créés et TC le temps de création pour des paquets de taille 1* ***alors*** *$\frac{EC}{u}$ est le nombre maximal d'états créés et $\frac{TC}{u^2}$ est le temps maximal de création pour des paquets de taille u.*

Nous insistons sur le fait que $\frac{EC}{u}$ et $\frac{TC}{u^2}$ sont des maxima car les équations 2 et 3 sont des bornes pour EC et TC.

Preuve: Supposons que EC et TC sont le nombre d'états créés et le temps de création (pour des paquets de taille 1) et prenons des paquets de taille u.

Remarquons tout d'abord que la prise de paquets de ressources n'influe en rien sur la taille du problème $\mathcal{N}$. De même, elle laisse invariants les nombres de niveaux dans une unité et le nombre de tâches par niveau.

En fait, prendre des paquets de taille u réduit en conséquence la "deadline" d'une unité. Au lieu d'être d_i, elle devient $\frac{d_i}{u}$.

Donc, l'équation (2) implique que $\frac{EC}{u}$ est le nombre maximal d'états qui seront créés. Et l'équation (3) implique que $\frac{TC}{u^2}$ est le temps de création maximal.

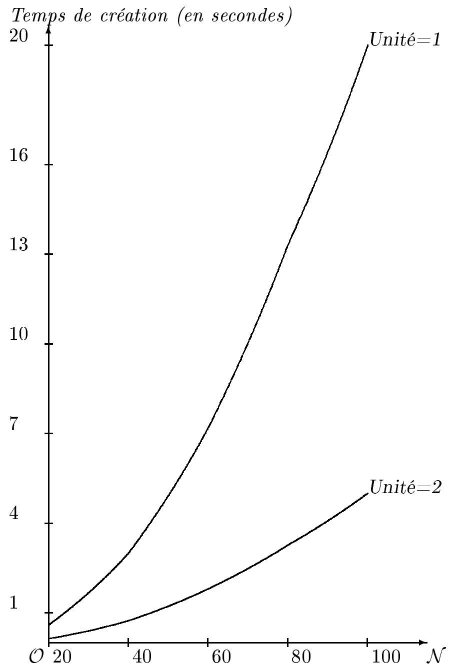
Le dernier terme de (3) peut-être modifier indirectement par les paquets mais il ne s'agit que d'une constante qui ne rentre pas en compte dans notre calcul de complexité. ■

5 Conclusion et perspectives

Cette approche dérivée de MDPC[7] permet de traiter un plus grand nombre de problèmes. En effet, MDPC et IS[6] ne traitent que des plans d'action linéaires ou les unités de raisonnement progressif sont linéaires. De plus, notre approche prend en compte un fort degré d'incertitude sur les ressources consommées à l'exécution, ce que ne prennent pas en compte *"imprecise computation"*[3, 4] et *"design-to-time"*[5]. Enfin, elle reprend une notion de dépendance (permet et interdit) vue dans *"Task interdependencies in design-to-time real-time scheduling"* [8].

Cependant, elle requiert beaucoup d'espace mémoire et de temps puisqu'elle construit tout le PDM et le stocke en mémoire (ce qui est à l'origine de la prise en compte du fort degré d'incertitude). Mais, l'utilisation de paquets de

FIG. 2: Temps de création pour des paquets de taille 1 et 2



ressources permet de maîtriser la taille du processus décisionnel de Markov pour une perte raisonnable sur la qualité (environ 4,5 % pour des paquets de taille 2, Cf. table 1). Pour finir, elle est peu dynamique puisque nous ne pouvons pas ajouter d'unités durant le traitement du plan d'action et après. Dans des travaux futurs, nous nous intéresserons à l'aspect plus dynamique, en nous inspirant des travaux déjà effectués avec l'"opportunity cost"[9].

Dans cet article, nous nous sommes restreints à la ressource temps. L'utilisation d'un vecteur de ressources permettrait de prendre en compte plus de types de ressources telles que le capacité mémoire et l'énergie. Ensuite, nous pourrions nous intéresser à l'incertitude sur le gain et nous pourrions étendre cette approche à des environnements plus dynamiques.

Dans les expérimentations (Cf. 4), l'URP utilisée pourrait être adaptée pour résoudre le problème lié au satellite (en orbite autour de la terre). Ainsi, il pourrait traiter un plus grand nombre d'unités. En ce moment, il ne traite que 20 unités alors que la demande est de 100.

Références

[1] E. J. Horvitz. Reasoning about beliefs and actions under computational resource constraints. In *Workshop UAI-87*, pages 429–444, 1987.

[2] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17 :73–83, 1996.

[3] D. Hull, W. C. Feng, and J. W. S. Liu. Operating system support for imprecise computation. In *AAAI Fall Symposium on Flexible Computation*, pages 96–99, 1996.

[4] J. Liu, K. Lin, W. Shih, J. Chung A. Yu, and W. Zhao. Algorithms sor scheduling imprecise computations. *IEEE Computer*, 24 :58–68, 1991.

[5] A. Garvey and V. Lesser. Desing-to-time real-time scheduling. *IEEE Transaction on systems, Man, and Cybernetics*, 23 :1491–1502, 1993.

[6] A-I. Mouaddib and S. Zilberstein. Handling duration uncertainty in meta-level control of progressive processing. In *IJCAI*, pages 1201–1206, 1997.

[7] A-I. Mouaddib and S. Zilberstein. Optimal scheduling of dynamic progressive processing. In *ECAI*, pages 449–503, 1998.

[8] A. Garvey, M. Humphrey, and V. Lesser. Task interdependencies in desing-to-time real-time scheduling. In *AAAI*, pages 580–585, 1993.

[9] S. Zilberstein and A-I. Mouaddib. Reactive control of dynamic progressive processing. In *IJCAI*, pages 1268–1273, 1999.