Une méthode inverse pour les plus courts chemins

Étienne ANDRÉ
LSV – ENS Cachan & CNRS, France
Etienne.Andre@lsv.ens-cachan.fr

***Résumé*—Nous considérons ici une adaptation de la *méthode inverse*, permettant de synthétiser des bornes de paramètres temporels garantissant un fonctionnement nominal dans le cadre des automates temporisés paramétrés, à deux algorithmes de la littérature : l'algorithme de Floyd–Warshall déterminant le plus court chemin dans un graphe orienté valué, et l'algorithme d'itération de politiques dans le cadre des processus de décision markoviens. Ces deux algorithmes permettent des applications aux systèmes temps réel, tels que la vérification de circuits.**

I. CONTEXTE

Nous décrivons ici une adaptation de la *méthode inverse* [3], initialement définie dans le cadre de la séparation temporelle d'évènements [6] puis étendue aux automates temporisés paramétrés. Les automates temporisés [1] sont des automates d'états finis équipés d'horloges, variables à valeurs réelles dont la valeur croît continûment. Les automates temporisés paramétrés [2] sont obtenus en autorisant dans les gardes et invariants l'usage, non plus de constantes, mais de *paramètres*. Un problème classique du cadre des automates temporisés paramétrés consiste à calculer une contrainte sur les paramètres du système, telle que, pour toute valuation des paramètres vérifiant cette contrainte, le système se comportera d'une façon définie comme *correcte*. Le *problème inverse* est le suivant : étant donné un automate temporisé paramétré $\mathcal{A}$ et une instance de référence des paramètres π_0, nous cherchons à obtenir une contrainte K_0 sur les paramètres vérifiée par π_0 telle que, pour toute instance π des paramètres vérifiant K (noté $\pi \models K$), le système instancié par π se comportera de manière *équivalente* au système de référence instancié par π_0. Cette notion d'équivalence entre les deux systèmes est en fait une *équivalence abstraction faite du temps* [3], c'est-à-dire que les traces d'exécution des deux systèmes, vues comme des séquences alternantes d'états de contrôle et d'actions, sont *identiques*. Cette méthode inverse permet donc la synthèse de bornes de paramètres temporels garantissant un fonctionnement nominal. Des applications à plusieurs protocoles de communication, à des circuits mémoire, ainsi qu'à des études de cas industrielles ont été effectuées [4].

Nous proposons ici une adaptation de cette méthode inverse à deux autres types de modèles : l'algorithme de Floyd–Warshall déterminant le plus court chemin dans un graphe orienté valué (section II), et l'algorithme d'itération de politiques dans le cadre des processus de décision markoviens (section III). Nous donnons pour chacun de ces deux algorithmes un exemple intuitif, ainsi que des éléments de motivation pour la vérification de systèmes. Nous donnons en conclusion des directions de recherches futures (section IV).

Ces travaux sont partiellement financés par l'Agence Nationale de la Recherche, projet ANR-06-ARFU-005, et par l'Institut Farman de l'ENS Cachan.

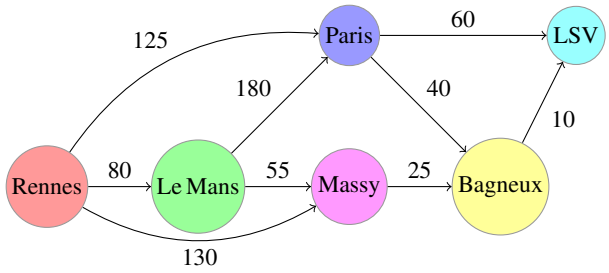

FIG. 1. Comment se rendre de Rennes à Cachan

II. PLUS COURT CHEMIN DANS UN GRAPHE VALUÉ

Nous nous intéressons dans cette section à l'algorithme de Floyd–Warshall [7], qui calcule le plus court chemin entre tous les couples de sommets d'un graphe orienté valué. On considère qu'un graphe orienté valué $\mathcal{G}$ est un couple (E, C), où E est un ensemble de sommets, et C une matrice de coûts à valeur dans $\mathbb{R}^+ \cup \infty$[1] telle que, pour tout $(i, j) \in E \times E$, C_{ij} représente le coût nécessaire pour passer de l'état i à l'état j. S'il n'existe pas de transition entre i et j, on fixe $C_{ij} = \infty$.

A. L'algorithme de Floyd–Warshall

Considérons l'exemple d'un doctorant souhaitant se rendre de Rennes au LSV, à Cachan, en transports en commun, et ce, aussi rapidement que possible. Les différentes options possibles, ainsi que la durée de chaque étape (exprimée en minutes) sont représentées sous la forme du graphe orienté valué décrit sur la figure 1. L'ensemble des sommets E est ici l'ensemble des villes, et la matrice C est décrite par les arcs valués entre les villes.

Appliquons l'algorithme de Floyd–Warshall tel que donné sur la figure 2. Cet algorithme calcule d'une part la matrice V des plus courts chemins, donnant la valeur du coût du plus court chemin entre deux sommets quelconques du graphe, et d'autre part la matrice des successeurs S, indiquant le sommet successeur de i sur le plus court chemin entre i et j, et ce pour tout couple de sommets (i, j). Une application de l'algorithme de Floyd–Warshall au graphe orienté valué de la figure 1 permet de déterminer que le trajet le plus court entre Rennes et Cachan est d'aller directement de Rennes à Massy, puis à Bagneux et enfin à Cachan (pour un total de 165 minutes).

[1] L'algorithme de Floyd–Warshall autorise en réalité les coûts négatifs, à condition de ne pas contenir de cycle de coût strictement négatif.

ALGORITHME Floyd–Warshall($\mathcal{G} = (E, C)$)
Entrée $\mathcal{G}$: Graphe orienté valué
Sortie V : Matrice des plus courts chemins
S : Matrice des successeurs

$V := C$
pour tout $k \in E$ **faire**
  pour tout $i \in E$ **faire**
    pour tout $j \in E$ **faire**
      $V_{ij} := \min(V_{ij}, V_{ik} + V_{kj})$
      $S_{ij} := \begin{cases} j & \textbf{si } V_{ij} \leq V_{ik} + V_{kj} \\ k & \textbf{sinon} \end{cases}$

FIG. 2. L'algorithme de Floyd–Warshall

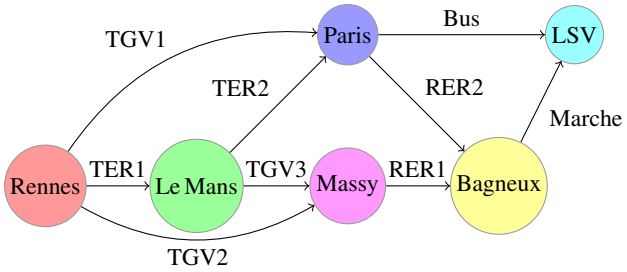

FIG. 3. Un exemple de graphe orienté valué paramétré

ALGORITHME *Inspeqtor*($\mathcal{G} = (E, D), \pi_0$)
Entrée $\mathcal{G}$: Graphe orienté valué paramétré
π_0 : Instance des paramètres
Sortie K_0 : Contrainte sur les paramètres
Variables V : Matrice des plus courts chemins
W : Matrice paramétrée des plus courts chemins
S : Matrice des successeurs

$V := D[\pi_0]$
$W := D$
pour tout $k \in E$ **faire**
  pour tout $i \in E$ **faire**
    pour tout $j \in E$ **faire**
      si $V_{ij} \leq V_{ik} + V_{kj}$ **alors**
        $S_{ij} := j$
      sinon
        $V_{ij} := V_{ik} + V_{kj}$
        $W_{ij} := W_{ik} + W_{kj}$
        $S_{ij} := k$
$K_0 :=$ `vrai`
pour tout $i \in E$ **faire**
  pour tout $j \in E$ **faire**
    pour tout $k \in E$ t.q. $k \neq S_{ij}$ **faire**
      $K_0 := K_0 \wedge \{W_{ij} \leq W_{ik} + W_{kj}\}$

FIG. 4. Algorithme *Inspeqtor*

B. Graphe orienté valué paramétré

Supposons maintenant que les temps de trajet sont tous fiables, à l'exception du trajet entre Massy et Bagneux, qui est suceptible d'être allongé en raison de divers incidents d'exploitation récurrents[2]. Dès lors, le doctorant souhaitant se rendre de Rennes à Cachan se pose la question suivante : jusqu'à quel retard sur le trajet Massy – Bagneux son trajet habituel restera-t-il le plus court ? S'il craint un retard plus important que le retard maximum, il pourra en effet reconsidérer ses habitudes et changer d'option (par exemple via Paris).

On considère alors un graphe orienté valué *paramétré*, c'est-à-dire un graphe orienté valué où les coûts ne sont plus des constantes, mais des *paramètres*. Chaque paramètre représente une constante réelle inconnue. On considère dans la suite un ensemble donné $P = \{p_1, \ldots, p_M\}$ de paramètres. Un *graphe orienté valué paramétré (GOVP)* $\mathcal{G}$ est un couple (E, D), où E est un ensemble de sommets, et D une matrice de coûts paramétrés à valeur dans $P \cup \infty$[3] telle que, pour tout $(i, j) \in E \times E$, D_{ij} représente le coût paramétré nécessaire pour passer de l'état i à l'état j. Si $D_{ij} = \infty$, on considère qu'il n'existe pas de transition entre i et j.

Par exemple, les différentes options possibles pour se rendre de Rennes à Cachan sont modélisées par le GOVP représenté sur la figure 3.

Soit $\mathcal{G} = (E, D)$ un GOVP. Pour une certaine instance $\pi = \{\pi_1, \ldots, \pi_M\}$ des paramètres du système, on note $D[\pi]$ la matrice des coûts (numériques) où chaque paramètre p_i a été remplacé par la valeur numérique π_i. Par extension, on note $\mathcal{G}[\pi]$ le graphe orienté valué (classique) $(E, D[\pi])$. Par exemple, soit $\mathcal{G}$ le GOVP représenté sur la figure 3. Soit π_0 l'instance suivante des paramètres du système (valeurs en minutes) :

$TGV1$ = 125	$TGV2$ = 130	$TGV3$ = 55
$TER1$ = 80	$TER2$ = 180	Bus = 60
$RER1$ = 25	$RER2$ = 40	$Marche$ = 10

Le graphe orienté valué $\mathcal{G}[\pi_0]$ est alors celui de la figure 1.

Le problème auquel on s'intéresse est le suivant : soit $\mathcal{G}$ un GOVP et π_0 une instance de référence des paramètres du système. On cherche à synthétiser une contrainte (c'est-à-dire une conjonction d'inégalités linéaires) K_0 sur les paramètres telle que :

1) $\pi_0 \models K_0$, et
2) pour tout $\pi \models K_0$, le chemin le plus court entre deux sommets du graphe $\mathcal{G}[\pi]$ est le même qu'entre ces deux sommets dans $\mathcal{G}[\pi_0]$.

C. L'algorithme Inspeqtor

Adaptons l'algorithme de Floyd–Warshall de la figure 2 afin de permettre la génération de la contrainte K_0. Une adaptation stricte de la méthode inverse [3] consisterait, à chaque modification de V_{ij} et S_{ij} (c'est-à-dire lorsque $V_{ij} > V_{ik} + V_{kj}$), à générer une inégalité calquée sur cette comparaison $V_{ij} > V_{ik} + V_{kj}$. Néanmoins, pour éviter d'obtenir une contrainte K_0 potentiellement trop forte, la génération des contraintes s'effectue dans une seconde série de boucles imbriquées effectuée après la première, c'est-à-dire une fois que la matrice S des successeurs est optimale.

Cet algorithme est présenté sur la figure 4. Il prend en entrée un GOVP $\mathcal{G} = (E, D)$ et une instance π_0 des paramètres, et retourne une contrainte K_0. La première série de boucles imbriquées calcule, comme dans l'algorithme classique de Floyd–Warshall, la matrice (instanciée) V des plus courts chemins entre deux sommets quelconques du graphe, ainsi

[2]Toute similitude avec une situation réelle concernant le RER B serait purement fortuite.

[3]On peut en fait étendre ce formalisme à une matrice de coûts paramétrés à valeur dans $P \cup \mathbb{R} \cup \infty$, c'est-à-dire que l'on autorise les coûts paramétrés ou constants.

que la matrice S des successeurs. En parallèle, la matrice W des plus courts chemins paramétriques est également calculée, sur la même base que V. Chaque élément W_{ij} contient une somme de coûts paramétriques correspondant au plus court chemin entre i et j. On notera que $V = W[\pi_0]$.

La seconde série de boucles imbriquées calcule la contrainte K_0. L'ajout de l'inégalité peut se résumer ainsi : pour calculer le chemin le plus court de i à j, pour tout sommet k différent du successeur optimal S_{ij} de i vers j, le chemin en passant par k est nécessairement plus long que celui passant par le successeur optimal S_{ij}[4]. Ainsi, cette contrainte garantit que le plus court chemin est bien celui calculé par l'algorithme classique de Floyd–Warshall.

D. Implémentation et applications

Cet algorithme a été implémenté sous la forme de l'outil INSPEQTOR (pour *INference of Shortest Paths with EQuivalent Time-abstract behaviOR*), un programme Caml de 3000 lignes. En appliquant cet outil au GOVP de la figure 3, ainsi qu'à l'instance π_0 mentionnée plus haut, nous obtenons la contrainte K_0 suivante[5] :

$$\begin{array}{rr} & RER2 + Marche \leq Bus \\ \wedge & TGV3 + RER1 \leq TER2 + RER2 \\ \wedge & TGV2 + RER1 \leq TGV1 + RER2 \\ \wedge & TGV2 \leq TER1 + TGV3 \\ \wedge & TGV1 \leq TER1 + TER2 \end{array}$$

En instanciant l'ensemble des paramètres, sauf $RER1$, par leur valeur de référence telle que définie dans π_0, nous obtenons l'inégalité suivante : $RER1 \leq 35$. Ainsi, tant que le trajet entre Massy et Bagneux dure moins de 35 mn (donc avec un retard inférieur à 10 mn), le trajet optimal entre Rennes et Cachan reste celui emprunté habituellement par notre doctorant (via Massy et Bagneux). En revanche, au-delà, la contrainte K_0 est violée, et aucune garantie n'est plus donnée sur le chemin le plus court entre deux points du graphe.

Nous avons ici considéré un exemple intuitif dans un souci de lisibilité ; une application directe de cette méthode est en fait l'étude de systèmes temporisés tels que les circuits. Dans ce cas, les composants (et câbles) d'un circuit sont considérés comme des éléments que l'électricité mettra un certain temps à traverser, certes faible, mais non négligeable dans le cadre de la vérification de tels systèmes. Dès lors, chaque composant du système peut être représenté par l'arc d'un graphe orienté valué. Il est alors intéressant de savoir si l'on peut changer un composant du système par un autre plus lent, dans un souci économique, et ce sans remettre en cause le fonctionnement global du système. En générant une contrainte grâce à l'outil INSPEQTOR, il est alors possible de changer certains composants du système, pour peu que leur nouvelle durée de traversée vérifie toujours la contrainte. Notons ici que l'algorithme *Inspeqtor* est adapté à une étude des circuits où les fonctionnalités des éléments ne sont pas prises en compte. En effet, le modèle des graphes orientés valués est adapté pour calculer une approximation du temps de réponse du circuit. Si l'on souhaite prendre en compte les fonctionnalités des éléments (portes « et », « ou », etc.) mais qu'il n'y a pas d'élément mémorisant, l'analyse de séparation temporelle des évènements, à laquelle on peut appliquer la méthode inverse définie dans ce cadre [6], est plus adaptée. Enfin, dans le cas où le circuit contient des éléments mémorisants et des boucles non bornées, en plus des portes dont on souhaite prendre en compte la fonctionnalité, le modèle des automates temporisés paramétrés, auquel la méthode inverse [3] est appliquée, est le plus adapté.

[4]L'ajout de l'inégalité à K_0 est quelque peu simplifié sur la figure 4 : en réalité, aucune inégalité n'est générée si l'un des deux membres est infini ; en outre, on génère, en fonction des cas, une inégalité large ou stricte.

[5]Temps de calcul inférieur à un centième de seconde sur un Intel Quad Core 3 GHz avec 3.2 Go de RAM.

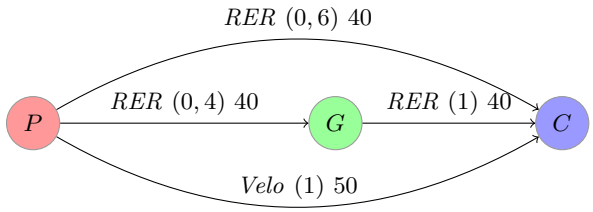

FIG. 5. Un exemple de processus de décision markovien

III. PROCESSUS DE DÉCISION MARKOVIENS

Nous considérons ici une extension de la méthode décrite dans la section II aux processus de décision markoviens.

A. Exemple

Un *processus de décision markovien (PDM)* [5] est un graphe orienté valué auquel on ajoute des labels (ou actions) sur les arcs. En outre, chaque arc porte une probabilité telle que, pour un sommet du graphe et une action donnés, la somme des probabilités quittant ce sommet via cette action soit égale à 1.

Considérons l'exemple d'un doctorant souhaitant se rendre de Paris à Cachan, soit en RER, soit à vélo. S'il prend le RER, celui-ci peut être fonctionnel, et atteindre Cachan en 40 mn comme à l'ordinaire avec une probabilité de $0,6$, ou bien être en panne avec une probabilité $0,4$, et n'être arrivé qu'à la moitié du trajet (à Gentilly) après 40 mn de trajet, pour ensuite atteindre Cachan en à nouveau 40 mn avec une probabilité 1. S'il opte pour le vélo, il arrivera de façon certaine (avec une probabilité 1) à Cachan en 50 mn. Le PDM de la figure 5 récapitule ces différentes options ; la notation « RER $(0,6)$ 40 » indique que, pour le label RER, cette transition peut être prise avec une probabilité $0,6$ pour un coût de 40.

Un problème classique des PDM est de déterminer une *politique optimale*, c'est-à-dire de résoudre le non-déterminisme en choisissant préalablement une action de sortie pour chaque sommet, et ce afin de minimiser le coût global du graphe. Un PDM dont le non-déterminisme a été résolu devient alors une *chaîne de Markov*, modélisable par exemple grâce à l'outil PRISM [8].

Dans notre cas, seul l'état P possède du non-déterminisme, puisqu'il est possible de choisir, soit l'action RER, soit l'action $Velo$. La question que se pose notre doctorant est la suivante : compte tenu des probabilités de panne, est-il plus intéressant d'opter pour le RER ou le vélo ?

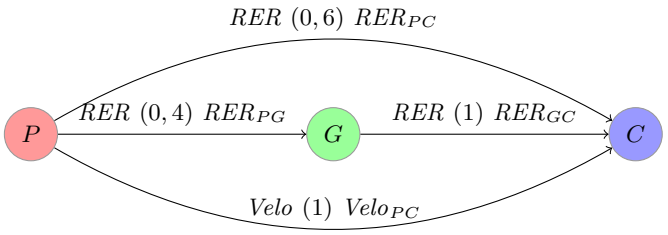

FIG. 6. Un exemple de processus de décision markovien paramétré

Une réponse classique à ce problème est d'utiliser un *algorithme d'itération de politiques* [9]. Sans rentrer dans le détail, cet algorithme part d'une politique quelconque puis, à chaque itération, calcule la valeur associée à chaque sommet pour cette politique grâce à un algorithme classique dit *d'itération de valeurs*, et améliore la politique en conséquence, et ce jusqu'à ce que celle-ci soit optimale. L'application d'un tel algorithme à notre exemple détermine que la politique optimale est d'utiliser le vélo, en 50 mn.

La nouvelle question que se pose notre doctorant est alors la suivante : jusqu'à quel allongement de son trajet à vélo celui-ci reste-t-il optimal ? En d'autres termes, sous quelle contrainte sur les temps de trajet la politique optimale de ce PDM restera-t-elle optimale ?

B. L'algorithme *Imperator*

Suivant un raisonnement similaire à celui de la section II, nous considérons des processus de décision markoviens *paramétrés*, c'est-à-dire où les coûts sont remplacés par des paramètres, de façon similaire au cas des GOVP. Étant donnés un PDM paramétré $\mathcal{P}$ et une instance π des paramètres, on note $\mathcal{P}[\pi]$ le PDM où les paramètres p_i ont été remplacés par la valeur π_i. Le PDM paramétré correspondant au PDM de la figure 5 est donné sur la figure 6, où le coût paramétrique RER_{PC} correspond à la durée en RER entre Paris et Cachan, et ainsi de suite pour RER_{PG}, RER_{GC} et $Velo_{PC}$. L'instance de référence π_0 est la suivante :

$$RER_{PC} = RER_{PG} = RER_{GC} = 40 \qquad Velo_{PC} = 50$$

L'algorithme *Imperator* prend en entrée un PDM paramétré $\mathcal{P}$ et une instance π_0 des paramètres. Il génère une contrainte K_0 sur les paramètres telles que, pour toute instance des paramètres vérifiant K_0, la politique optimale de $\mathcal{P}[\pi]$ sera identique à la politique optimale de $\mathcal{P}[\pi_0]$.

Le principe général de cet algorithme est le suivant :

1) calculer la politique optimale pour $\mathcal{P}[\pi_0]$;
2) calculer la valeur associée à chaque état grâce à un algorithme classique d'itération de valeurs ;
3) calculer la valeur paramétrique associée à chaque état grâce à un algorithme paramétré d'itération de valeurs ;
4) pour tout sommet s, pour toute action a différente de l'action optimale a_{opt} pour s (c'est-à-dire donnée par la politique optimale), générer une contrainte indiquant que la somme des valeurs paramétriques des états atteints depuis s par a, pondérée par les probabilités respectives, est supérieure à celle des états atteints depuis s par a_{opt}.

Cet algorithme a été implémenté sous la forme de l'outil IMPERATOR (pour *Inverse Method for Policy with Reward AbstracT behaviOR*), un programme Caml de 4300 lignes. En appliquant cet outil à l'exemple de la figure 5, nous obtenons la contrainte K_0 suivante[6] :

$$3RER_{PC} + 2RER_{PG} + 2RER_{GC} \geq 5\,Velo_{PC}$$

En instanciant tous les paramètres sauf $Velo_{PC}$ par leur valeur définie dans π_0, on obtient l'inégalité $Velo_{PC} \leq 56$. Par conséquent, si notre doctorant met moins de 56 mn à vélo, la politique consistant à opter pour le vélo reste la politique optimale. Au-delà, K_0 est violée et aucune garantie n'est donnée sur la politique optimale.

On remarque que ce résultat pouvait se prédire sur un exemple aussi simple, puisqu'il est aisément visible sur la figure 5 que le temps moyen de trajet en RER est de $0,6 * 40 + 0,4 * (40 + 1 * 40) = 56$ mn. En revanche, il est intéressant d'utiliser la contrainte générée IMPERATOR pour des systèmes plus complexes. Pour un système avec 11 états et 132 transitions, correspondant à la modélisation du parcours d'un robot dans un espace physique borné, une contrainte est générée en $0,17$ seconde par IMPERATOR.

IV. REMARQUES FINALES

Nous avons présenté ici une adaptation de la méthode inverse à deux algorithmes de la littérature traitant des plus courts chemins dans un cadre probabiliste ou non. Nous travaillons en ce moment sur une adaptation à d'autres algorithmes.

En particulier, l'adaptation à un algorithme d'itération de politiques pour les processus de décision markoviens à deux coûts est à l'étude. Cette présence de deux coûts distincts permet ainsi de modéliser la gestion de puissance dynamique (*dynamic power management* [10]) des systèmes temps réel, où l'on souhaite par exemple minimiser l'énergie tout en gardant les pertes de requêtes inférieures à une certaine valeur.

RÉFÉRENCES

[1] R. Alur and D. L. Dill. A theory of timed automata. *TCS*, 126(2) :183–235, 1994.
[2] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC '93*, pages 592–601, New York, USA, 1993. ACM.
[3] É. André, T. Chatain, E. Encrenaz, and L. Fribourg. An inverse method for parametric timed automata. *International Journal of Foundations of Computer Science (IJFCS)*. To appear.
[4] É. André, E. Encrenaz, and L. Fribourg. Synthesizing parametric constraints on various case studies using IMITATOR. Research report, LSV, ENS Cachan, France, June 2009.
[5] R. Bellman. A Markov decision process. *Journal of Mathematical Mechanics*, 6 :679–684, 1957.
[6] E. Encrenaz and L. Fribourg. Time separation of events : An inverse method. In *Proceedings of the LIX Colloquium '06*, volume 209 of *ENTCS*, Palaiseau, France, 2008. Elsevier Science Publishers.
[7] R. W. Floyd. Algorithm 97 : Shortest path. *Commun. ACM*, 5(6), 1962.
[8] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM : A tool for automatic verification of probabilistic systems. In *TACAS'06*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
[9] R. A. Howard. *Dynamic Programming and Markov Processes*. John Wiley and Sons, Inc., 1960.
[10] G. A. Paleologo, L. Benini, A. Bogliolo, and G. De Micheli. Policy optimization for dynamic power management. In *DAC '98*, pages 182–187, New York, NY, USA, 1998. ACM.

[6] Temps de calcul inférieur à un centième de seconde sur un Intel Quad Core 3 GHz avec 3.2 Go de RAM.