Vers une planification en environnement dynamique

Emmanuel Guere, Rachid Alami

▶ **To cite this version:**

Emmanuel Guere, Rachid Alami. Vers une planification en environnement dynamique. RFIA : Reconnaissance des Formes et Intelligence Artificielle, 2000, Paris, France. hal-01979814

HAL Id: hal-01979814
https://laas.hal.science/hal-01979814

Submitted on 13 Jan 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Vers une planification en environnement dynamique

Towards planning in dynamic environment

Emmanuel Guéré[1] Rachid Alami[1]

[1] LAAS-CNRS
7, avenue du colonel Roche
31077 Toulouse cedex 4
France
{guere, rachid}@laas.fr

Résumé

Pour planifier une mission en environnement dynamique, un robot doit pouvoir raisonner de manière explicite sur l'incertitude, le non-déterminisme et les évolutions contingentes de l'environnement. Dans cet article, nous proposons une représentation de l'action et du changement et un algorithme qui répondent à ces exigences. Nous définissons deux sources de non-déterminisme dans l'issue d'une action, l'une due à une modélisation insuffisante, non-déterminisme de modélisation, et l'autre due à un manque d'informations sur l'état du monde, non-déterminisme d'incertitude. Ensuite nous définissons les événements contingents qui permettent de prévoir correctement (dans la limite du modèle) les effets de bord de certaines actions ou inactions. Nous proposons enfin un algorithme implémenté pour la recherche de plan. Le plan ainsi obtenu, qui peut être conditionnel grâce aux actions de perception qui lèvent certaines indéterminations, est qualifié en termes de nécessité de réussite ce qui permet au planificateur de ne développer que les branches nécessaires. Nous garantissons tout de même sa robustesse par l'introduction d'états stables et d'états sûrs.

Mots Clef

Planification, raisonnement probabiliste ou possibiliste.

Abstract

To plan a mission in a dynamic environment, a robot needs to reason explicitly on uncertainty, non-determinism and environment contingencies. This paper proposes an action representation and a world model and an algorithm to deal with inaccuracy and possible evolutions. We distinguish between two types of non-determinism in action outcomes: a non-determinism from insufficient modeling and a non-determinism from uncertainty. We also define contingent events to correctly foresee actions or inactions results in a dynamic environment. Finally we present an implemented planner. We qualify plans with a goal achievement necessity which allows the planner to explore only necessary branch (note that perception actions create conditional plan by removing uncertainties). The introduction of safe and stable state ensures the plan robustness.

Keywords

Planning, probabilistic or possibilistic reasoning.

1 Introduction

Un robot autonome opère dans un environnement imparfaitement connu et en évolution permanente. Pour mener à bien sa mission, le robot va planifier un ensemble de séquences de tâches à réaliser. La robustesse du plan dépendra donc de la capacité du robot à raisonner de manière explicite sur les changements de l'environnement afin d'en profiter ou d'éviter les situations indésirables.

La principale difficulté provient de l'application d'une action dans un contexte incertain et partiellement observable. Nous distinguons dans le paragraphe 2.3 deux cas entraînant des effets non-déterministes, soit par une modélisation «trop grossière», soit par une incertitude sur les «préconditions» des actions.

Un point clef de la planification sous incertitude réside dans l'habileté à explorer les cas les plus «probables». Nous nous aidons ici de la théorie des possibilités [6] pour obtenir une mesure qualitative des apparitions possibles d'effets permettant à la fois une représentation plus fidèle de la réalité ainsi qu'une évaluation qualitative du plan trouvé; ainsi, on simplifie la re-

cherche de plans en laissant de côté les cas très peu «probables» (par rapport à la qualité voulue du plan). Par ailleurs, l'introduction des situations interdites (e.g. dangereuses pour le robot) et des événements contingents nous incite à définir une notion de robustesse du plan (grâce aux états sûrs) pour ces cas ainsi que pour le but.
Le formalisme STRIPS [8], souvent utilisé, ne permettant pas de prendre en compte l'incertitude et la dynamique de l'environnement, nous allons donc, dans cet article, présenter une nouvelle modélisation du monde et des opérateurs. Dans un premier temps, nous étudierons la notion d'incertitude en planification et exposerons notre modélisation suivie d'une comparaison avec la littérature. Ensuite nous discuterons des difficultés liées aux événements contingents, pour enfin terminer par une présentation globale de l'algorithme employé qui correspond à une extension de celui présenté dans [9].

2 Une connaissance incertaine de l'environnement

2.1 Décor de l'exemple

Afin d'illustrer le propos, prenons un exemple simple de robot cambrioleur de musée. Un robot volant veut dérober un tableau qui se trouve dans une salle de musée sécurisée. Il sait qu'un contact avec le plancher ferme la porte d'entrée (par laquelle il ne passe pas, mais seule entrée possible pour les policiers), que les tableaux du musée sont souvent protégés par un capteur ... Nous décrirons, par la suite en fonction de nos besoins, les opérateurs disponibles, les événements contingents susceptibles d'arriver et les états interdits.

2.2 Notions de base

Nous rappelons, dans les paragraphes qui suivent, les définitions nécessaires à une bonne compréhension de l'article.

Théorie des possibilités. La théorie des possibilités [18, 6] offre un modèle pour l'incertain ; chaque fait A est associé à deux valeurs :

- $\Pi(A)$: la possibilité que le fait A soit vrai,
- $N(A)$: la nécessité que le fait A soit vrai.

Une relation de dualité permet de relier A avec son opposé $\neg A$ par la relation $N(A) = 1 - \Pi(\neg A)$. Par exemple si le robot ne sait pas s'il y a un capteur au niveau du tableau ou non, nous écrivons : $\Pi(capteur) = 1$ et $N(capteur) = 0$. Par contre si le robot savait qu'il y a souvent un capteur, nous écririons : $\Pi(capteur) = 1$ et $N(capteur) = 0,7$. Notons que les mesures de possibilité et de nécessité sont des mesures qualitatives (seul l'ordre entre les différentes valeurs compte).

Définition et application d'une action STRIPS. Notre modèle d'action est en fait une extension de celui proposé par [8] : soit une action A, P ses préconditions (conjonction[1] de faits $\{p_i\}$) et E ses effets. E peut se décomposer en E^+ (liste d'ajouts $\{e_i^+\}$) et en E^- (liste de retraits[2] $\{e_i^-\}$). Pour que l'action A soit applicable dans l'état S ($\{s_i\}$), il faut que ses préconditions soient vérifiées ($P \subseteq S$). Dans ce cas le nouvel état S' devient $(S - E^-) \cup E^+$.

Les actions conditionnelles. Les effets d'une action conditionnelle sont dépendants du contexte : suivant le contexte (l'état S dans lequel l'action A est appliquée) les effets peuvent être différents. Une action conditionnelle A s'exprime de la manière suivante : soit P ses préconditions, $\{Cond_i\}_{i \in 1..n}$ un ensemble de n conditions mutuellement exclusives et $\{E_i\}_{i \in 1..n}$ l'ensemble des n effets associés. Si nous appliquons A dans l'état S contenant $Cond_i$, les effets seront E_i. Evidemment toute action conditionnelle peut être scindée en 2^{nm} actions[3] avec m le nombre maximal de conjonctions dans les conditions $Cond_i$.

Les actions non-déterministes. Nous disons d'une action A qu'elle est non-déterministe si elle est définie par un ensemble d'issues E_i et qu'il est impossible de prévoir avec certitude laquelle sera utilisée au moment de l'exécution. Une action non-déterministe peut donc être définie par des préconditions P et des issues E_i correspondant aux différents effets pouvant survenir à l'exécution. Notons que la description des issues E_i est exhaustive et qu'une seule d'entre elles apparaîtra.

2.3 Les origines du non-déterminisme

Comme nous l'avons mentionné en introduction, nous distinguons deux sources de non-déterminisme.

Tout d'abord le non-déterminisme dû à une représentation trop «grossière»[4] de l'environnement. Par exemple pour modéliser l'action *Prendre(T:Tableau;*

1. Pour simplifier, nous allons les représenter par des ensembles.
2. Notons que $E^- \subseteq P$.
3. [10, 1] proposent une modélisation compactée des actions conditionnelles sous Graphplan.
4. Cette représentation trop «grossière» peut être due à une incapacité à modéliser plus finement (e.g. modéliser le résultat d'un jeté de dés) ou à une volonté de simplifier le modèle.

L:Lieu)[5] nous pouvons écrire :

si	le robot est en L, qu'il ne tient rien, et que le tableau T est en L,
alors	**soit** le robot tient T, **soit** le robot déclenche l'alarme et T disparaît.

Dans ce cas nous savons qu'il existe deux issues possibles pour l'action, mais nous ne pouvons pas de manière précise distinguer celle qui s'exécutera. Donc après exécution d'une telle action nous avons deux mondes possibles : un où le robot déclenche l'alarme et un dans lequel le robot ne déclenche pas l'alarme[6].
Si nous diminuons la granularité de la représentation (ajout de la connaissance d'un capteur), nous pouvons transformer l'action non-déterministe en une action conditionnelle :

si	le robot est en L, qu'il ne tient rien, et que le tableau T est en L,	
alors	**s'il**	n'y a pas de capteur en L
	alors	le robot tient T,
	s'il	y a un capteur en L
	alors	le robot déclenche l'alarme et T disparaît.

Prenons A une action conditionnelle. Si l'état S est incertain ($S = S_1$ ou $S = S_2$ avec $S_1 = S' \cup Cond_i$ et $S_2 = S' \cup \neg Cond_i$), alors l'issue de l'exécution de A sera soit E_i soit E_j (avec $Cond_j = \neg Cond_i$), ce qui correspond à un résultat non-déterministe. Dans l'exemple précédent, bien que nous ayons rendu l'action déterministe en augmentant notre modèle, si l'on ne possède pas d'informations certaines sur *capteur* le résultat de l'exécution sera non-déterministe.
Nous définissons donc deux types de non-déterminisme : un premier dit de *modélisation* généré au moment de la création de la représentation de l'environnement, et un second dit *d'incertitude* généré par un manque d'information au moment de la planification. Nous remarquons donc qu'en environnement incertain, l'action conditionnelle correspond à une extension de l'action non-déterministe.

2.4 La modélisation de l'incertain

Les représentations usuelles. Nous pouvons distinguer plusieurs manières de représenter l'incertitude de la connaissance : l'ignorance totale d'un fait (e.g. le robot ne sait pas du tout s'il y a un capteur ou non), l'incertitude qualitative (le robot sait seulement qu'il y a souvent un capteur) et l'incertitude quantitative (le robot sait que dans 90% des cas il y a un capteur). La plupart des planificateurs travaillant avec des mondes incertains utilisent une disjonction de mondes possibles dès l'état initial (e.g. CASSANDRA [14], CGP [15] ...) ce qui correspond à notre idée de l'ignorance totale. BURIDAN [11] utilise les probabilités qui s'avèrent très utiles pour modéliser l'incertitude sur les transitions entre les états, mais la théorie des probabilités ne permet pas de modéliser l'ignorance totale. De plus, il existe de nombreuses situations dans lesquelles il est impossible de donner au robot des mesures statistiques. La théorie des possibilités présente le double intérêt de permettre de modéliser l'ignorance totale et de rester dans le domaine du qualitatif.

Notre représentation: deux niveaux d'incertitude. Comme nous venons de l'expliquer, nous allons utiliser la théorie des possibilités pour modéliser l'environnement. Un état est défini par une conjonction de faits possibilistes (i.e. une possibilité et une nécessité sont associées à chaque fait). Nous allons étendre l'*hypothèse du monde clos*[7] généralement utilisée avec le formalisme STRIPS. Nous étendons cette hypothèse par : *seuls les faits dont la nécessité est strictement positive sont représentés.* Le problème induit sur les négatifs sera réglé, à terme, par l'ajout de règles de cohérence. Cette hypothèse présente l'intérêt de réduire l'état du monde aux connaissances du robot ce qui permet de mieux cibler les faits intéressants[8].
Un exemple de représentation est proposé dans le tableau 1. Par contre, le but est modélisé par une conjonction de faits sans nécessité et une nécessité globale avec laquelle le but doit être atteint; par exemple, le but de notre robot cambrioleur est $Tient(Tableau) \wedge \neg Abimé(Tableau)$ et il doit le réaliser dans un premier temps avec une nécessité $N_{But} \geq 0,7$.

Fait	Nécessité
Est_en(Robot,L)	1
Est_en(Tableau,L)	1
¬ Abimé(Tableau)	1
¬ Alarme	1
¬ Police	1
Capteur	0,7
...	...

TAB. 1 – *Un exemple d'état initial*

Ce premier niveau d'incertitude permet de modéliser les différentes possibilités au sein d'un état de manière condensée. En effet un état S contenant n faits incertains ($N(F_i) < 1$ et $\Pi(F_i) > 0$) modélise en réalité

5. Prendre le tableau T qui se trouve en L.
6. Nous ne considérons, pour l'instant, aucun événement contingent.
7. Les faits qui sont faux ne sont pas représentés. Cette hypothèse permet de réduire considérablement la taille des bases de connaissances; en effet si on a $On(A,B)$ il faut aussi avoir $\neg On(B,A)$, $\neg On(B,C)$, $\neg On(A,C)$ ...
8. Si F n'est pas représenté, c'est que $N(F) = 0$ et $N(\neg F) = 0$, ce qui revient à dire que $\Pi(F) = 1$ soit l'ignorance totale sur F. Ce formalisme permet de cacher les faits inutiles sans pour autant les éliminer: ils restent accessibles par des actions de perception si elles existent. Une élimination efficace de faits est présentée dans [12].

2^n états distincts. Bien que très utile, cette représentation ne suffit pas pour distinguer les différentes issues des actions non-déterministes : l'issue de l'application de l'action non-déterministe *Prendre*, dans un état S certain avec le fait *alarme* faux, pourrait s'écrire $\Pi(Tient(T)) = 1$, $N(Tient(T)) = 0$, $\Pi(Alarme) = 1$ et $N(Alarme) = 0$; dans ce cas nous perdons l'information $Tient(T)$ est vrai quand *Alarme* est faux. Ce second niveau d'incertitude est en réalité conservé lors de l'expansion du graphe des états (approche Graphplan).

Approche Graphplan. Comme nous l'avons mentionné, nous utilisons une algorithmique inspirée de Graphplan [2]. Un des avantages de cette approche vient de la compression des états possibles à une étape donnée : les faits, même s'ils sont présents dans plusieurs états envigeasables à l'étape t, ne sont représentés qu'une seule fois. La cohérence et la reconstruction des états sont garanties par des règles d'exclusion.

Pour conserver cet avantage, nous fusionnons [9] les différentes issues possibles à chaque étape (un fait possibiliste n'apparait qu'une seule fois par étape), mais notons tout de même leurs provenances : lors de la recherche de plan, le planificateur utilisera ces informations pour reconstruire les issues.

Les états interdits. Un état interdit est une conjonction de faits ($\{F_i\}, F_i$ non-possibilistes) qui ne doivent jamais être vrais en même temps. Ces états sont utilisés pour modéliser les situations dangereuses pour le robot ou l'environnement (e.g. pour le robot cambrioleur, un état contenant à la fois la police et le robot dans le musée est un état dangereux pour son avenir...). Durant la planification ces états ne doivent jamais apparaître même avec une très faible possibilité!

2.5 Les actions, sources d'incertitudes?

La connaissance de l'environnement étant incertaine, nous devons définir un formalisme au niveau des actions qui permette de prendre en compte les différents cas possibles et d'y associer leurs effets (issues pour les cas conditionnels et non-déterministes).

Des actions possibilistes. Nous étendons dans cet article les actions classiques décrites en 2.2. Etant donné un état S ($\{(s_i, N(s_i))\}$) et une action déterministe A ayant comme précondition P ($\{p_i\}$) et les effets E ($\{(e_i, N(e_i|P))\}$ avec e_i pouvant être positif ou négatif), l'état résultant S' de l'application de A à S peut être calculé par la règle de propagation :

$$N^{S'}(e_i) \geq max(N^S(e_i), min(N(p_1), \ldots, N(p_n), N(e_i|P)))$$

avec $N(e_i|P)$ la nécessité que e_i soit vrai sachant que les préconditions sont vérifiées et $N^{S'}(e_i)$ la nécessité de e_i dans l'état S' (voir figure 1).

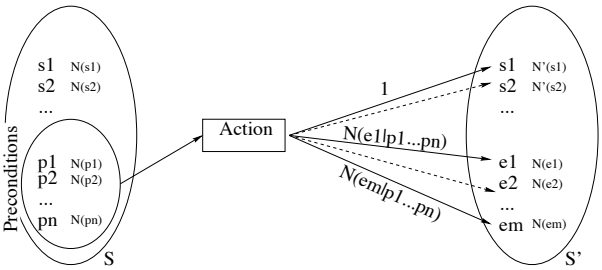

FIG. 1 – *Modèle d'action possibiliste*

Quand peut-on appliquer une action? Nous remarquons avec cette définition que si $N(e_i|P) < 1$ alors l'issue de l'exécution de l'action est incertaine même si les préconditions sont toutes totalement vraies. Prenons comme hypothèse [10] que seuls les effets décrits sont connus, c'est à dire que les effets autres ont une nécessité nulle et une possibilité de 1.

Le problème de l'applicabilité d'une action peut se traiter de deux façons. La première prend en considération le problème dans son ensemble (modèle symbolique, numérique, géométrique ...), le problème est en partie résolu par simulation : l'action instanciée $A(I_1)$ est normalement appliquée à l'état S_1, la simulation de l'exécution permet de connaître la façon dont l'action va être exécutée et ainsi de pouvoir simuler $A(I_1)$ dans l'état S_2 où les préconditions sont fausses (la figure 2 présente un exemple topologique 2D). Cette approche est intéressante mais trop coûteuse pour la phase de planification. La seconde solution consiste à considérer que toutes les issues non-déterministes envisageables sont définies par le programmeur dans les opérateurs. Rappelons, en effet, que toute action ayant n préconditions peut se transformer en une action conditionnelle avec 2^n cas! Dans cet article nous considérons seulement la seconde approche qui permet de mieux contrôler l'applicabilité des actions.

9. Si le fait F peut être obtenu de deux manières différentes, nous prenons pour nécessité de F dans la nouvelle étape la nécessité précédente la plus faible, l'hypothèse la plus pessimiste.

10. Nous pouvons démontrer que sous cette hypothèse simplificatrice la nécessité de l'effet e_i ne peut qu'augmenter. Le plus simple étant tout de même, dans ce cas là, de considérer qu'il ne se passe rien.

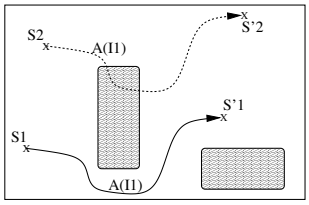

FIG. 2 – *Exemple d'application d'une action dans le mauvais état. Le robot a planifié sa trajectoire (action instanciée $A(I_1)$) en pensant être en S_1. Le trait pointillé correspond à l'application de $A(I_1)$ à l'état S_2 au lieu de S_1.*

Actions conditionnelles. Une action conditionnelle possibiliste (figure 3) est donc définie par :

Nom(Paramètres typés)		
Préconditions:	$p_1 \dots p_n$	
Effets:	$Cond_1$:	$(e_1^1, N(e_1^1\|P)) \dots (e_{l_1}^1, N(e_{l_1}^1\|P))$
	...	
	$Cond_m$:	$(e_1^m, N(e_1^m\|P)) \dots (e_{l_1}^m, N(e_{l_m}^m\|P))$

L'action *déconnecter* les capteurs (s'il y en a) du tableau s'écrit donc :

Déconnecter(T:Tableau; L:Lieu)		
Préconditions:	Est_en(Robot,L), Est_en(T,L)	
Effets:	*capteur*:	$(\neg capteur, 1)$
	$\neg$*capteur*:	*(Abimé(T),*0,7)

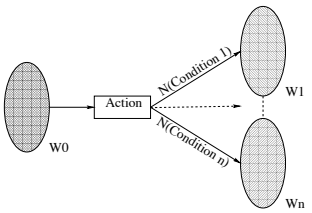

FIG. 3 – *Modèle d'action conditionnelle*

Actions non-déterministes. Une action non-déterministe possibiliste (figure 4) est donc définie par :

Nom(Paramètres typés)		
Préconditions:	$p_1 \dots p_n$	
Effets:	$(w_1, \Pi(w_1), N(w_1))$:	$(e_1^1, N(e_1^1\|P)) \dots (e_{l_1}^1, N(e_{l_1}^1\|P))$
	...	
	$(w_m, \Pi(w_m), N(w_m))$:	$(e_1^m, N(e_1^m\|P)) \dots (e_{l_1}^m, N(e_{l_m}^m\|P))$

Nous pouvons ainsi définir l'action *tester* qui teste s'il y a un capteur sur le tableau (dans notre exemple, nous utiliserons la version conditionnelle) :

Tester(T:Tableau; L:Lieu)		
Préconditions:	Est_en(Robot,L), Est_en(T,L)	
Effets:	$(w_1, 1, 0.7)$:	*Alarme*
	$(w_2, 0.3, 0)$:	$\neg$*Alarme*

Comme nous l'expliquions en 2.3 une action non-déterministe est une action «conditionnelle» dont les conditions ne sont pas définissables (parties non-modélisées du contexte) avec la granularité de la représentation, c'est pourquoi nous avons $(w_m, \Pi(w_m), N(w_m))$ au lieu de $Cond_m$ qui contient implicitement sa possibilité et sa nécessité dans l'état S. Evidemment nous pouvons combiner les actions conditionnelles avec les actions non-déterministes. Notons tout de même que les issues possibles doivent être exhaustives.

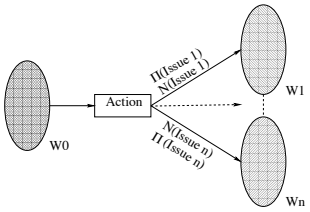

FIG. 4 – *Modèle d'action non-déterministe*

Actions de perception. Nous pouvons définir une action de perception (figure 5) par :

Nom?(Paramètres typés)		
Préconditions:	$p_1 \dots p_n$	
Effets:	$(w_1, \Pi(E_1), N(E_1))$:	$(e_1^1, 1) \dots (e_{l_1}^1, 1)$
	...	
	$(w_m, \Pi(E_m), N(E_m))$:	$(e_1^m, 1) \dots (e_{l_1}^m, 1)$

La seule action de perception que nous utilisons dans notre exemple se définit par :

Alarme?()		
Préconditions:	Est_dans(Robot,musée)	
Effets:	$(w_1, 1, 0)$:	$(Alarme, 1)$, $(Capteur, 1)$
	$(w_2, 1, 0)$:	$(\neg Alarme, 1)$, $(\neg Capteur, 1)$

Notons que notre planificateur n'est pas capable de déduire que le fait *Alarme* est corrélé avec le fait *Capteur*, nous simulons donc cette déduction dans notre action de perception.

Cette représentation pourrait être confondue avec celle des actions non-déterministes, mais il existe une grosse différence (outre le '?' qui permet de les distinguer sur papier) au niveau de la planification. En effet après l'exécution d'une action à effets non-déterministes, il est impossible de savoir de manière certaine quelle en est l'issue, tandis que l'issue d'une action de perception est connue à l'exécution. Mais il faut tout de même prévoir tous les cas possibles au niveau de la planification car il est impossible d'avoir de certitude avant l'exécution.

Les actions de perception permettent au robot d'améliorer sa connaissance sur un ensemble de faits. Dans l'implémentation courante, nous supposons que la perception est parfaite (i.e. $N(e_i|P) = 1$). Mais l'utilisation du conditionnel peut aider à modéliser une action

de perception qui échoue comme dans C-BURIDAN [5].

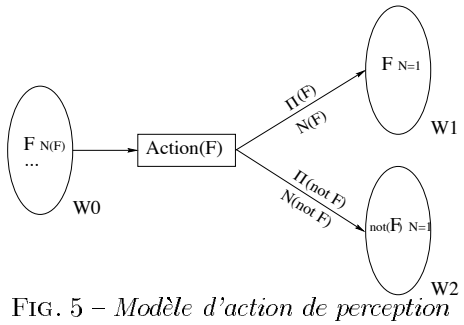

FIG. 5 – *Modèle d'action de perception*

2.6 Définition d'un plan

L'approche adoptée par [15] dans CGP consiste à trouver un plan qui est valide dans tous les cas possibles. Notre but consiste à qualifier la nécessité N_{But} [11] avec laquelle nous voulons que le but soit vrai, pour limiter la recherche à des plans plus simples s'ils existent.

Soit l'état initial E_{init}, le Plan $\mathcal{P}$ et l'état but E_{But}, l'application de $\mathcal{P}$ à E_{init} mène à E_{But} et $N(E_{But}) \geq N_{But}$ [12]. L'introduction de l'incertain permet de distinguer deux aspects d'une portion de plan utilisant plusieurs états possibles : l'aspect «plan conformant» [13] où une action s'applique à plusieurs états à la fois sans pouvoir préciser dans lequel le robot se trouve, et l'aspect «normal» dans lequel l'action est appliquée à un état connu (voir l'exemple de la figure 6).

On pourrait croire, en voyant l'arborescence d'un plan que l'on peut distinguer les parties «plan conformant» des parties «normales», mais comme nous le montre la figure, une action de perception ne discerne pas forcément tous les états possibles (e.g. les états w_{13} et w_{23}). Nous pouvons donc continuer à appliquer des actions à plusieurs états à la fois après une action de perception. Notons que toutes les branches de l'arbre doivent mener au but (avec une nécessité suffisante), car durant la phase de planification, le robot est incapable de connaitre le résultat de l'exécution d'une observation. Bien entendu, aucun état créé par l'application de $\mathcal{P}$ à E_{init} ne doit correspondre à un état interdit. Pour cela nous devons nous assurer que chaque action A_t de $\mathcal{P}$ est applicable dans tous les états S_t possibles et engendrer tous les états S_{t+1} même de très faible possibilité.

11. Il serait intéressant de pouvoir traiter les buts flexibles (qualifier indépendemment les faits de l'état but) en utilisant [7] ce que nous ne faisons pas pour l'instant.

12. Nous ne cherchons donc pas à maximiser la nécessité du plan comme c'est le cas dans [4].

13. *Conformant planning* en anglais.

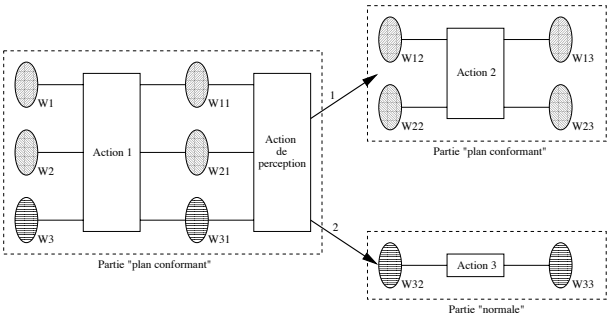

FIG. 6 – *Exemple de plan ayant une observation et des parties «plan conformant». Lorsque le planificateur ne peut pas (ou ne veut pas) distinguer les différents états possibles (w_1, w_2 et w_3) entre eux, il est doit appliquer une action commune (e.g. $Action_1$). Nous nommons ces portions de plan, des «plans conformants». Remarquons que l'application d'une action de perception, bien que source d'information, peut conserver des portions de «plans conformants».*

2.7 Expressivité de notre modèle

Dans cette partie nous allons comparer notre modèle du monde et des opérateurs à trois systèmes représentatifs.

BURIDAN, PGraphplan. PGraphplan [3] est un planificateur probabiliste basé sur Graphplan et sur les processus de décision de Markov. A tout instant, le robot sait avec certitude dans quel état il se trouve ; le non-déterminisme provient seulement (comme c'est le cas pour BURIDAN[11]) des actions dont les effets sont pondérés par une distribution de probabilité. Outre la différence probabilité/possibilisté, le modèle d'action de BURIDAN et de PGraphplan correspond, dans notre cas, à une action de perception ayant des effets de bord (i.e. l'action ne fait pas que donner de l'information, elle modifie aussi les faits de l'état) ; les probabilités n'étant présentes que sur les transitions entre états, elles correspondent à notre idée des $N(e_i|P)$.

POSPLAN. POSPLAN est un planificateur possibiliste présenté dans [4]. Un état est défini par une conjonction exhaustive de faits positifs ou négatifs (pas d'utilisation de l'hypothèse du monde clos, étendue ou non). L'incertitude est décrite au niveau des états par un ensemble d'états possibles pondérés par une mesure de possibilité (ce qui correspond à notre second niveau d'incertitude [14]). Les actions modélisées sont conditionnelles et comportent des transitions possibilistes ; la différence d'expressivité de ce modèle comparé au

14. Notons que cette représentation (pour nous le second niveau d'incertitude) est plus riche que notre premier niveau d'incertitude, mais plus coûteuse

notre provient seulement de l'observabilité totale des états de POSPLAN, il n'y a donc pas de problème d'applicabilité d'action et de «plan conformant».

CASSANDRA, CGP. CASSANDRA [14] est un planificateur capable de modéliser des effets conditionnels et non-déterministes. Un état est défini par une conjonction de faits positifs ou négatifs (utilisant l'hypothèse du monde clos). L'incertitude est introduite par une disjonction d'états initiaux possibles (i.e. pas de mesure d'incertitude, ce qui correspond à notre idée de l'ignorance totale). CGP [17] est un planificateur dérivé de Graphplan utilisant le même modèle et incluant les observations. Les observations sont définies sur la base d'actions conditionnelles (définies dans [15]) dont les effets peuvent être non-déterministes. Une proposition $K\neg u : v$ permet efficacement de noter les exclusions entre états possibles après observation. Par contre, CGP ne permet pas de construire de nouveaux mondes incertains (tous sont définis dans l'état initial) et de qualifier ses états ce qui l'oblige à être plus stricte durant les parties de «plan conformant».

3 Les événements contingents

3.1 Définition

Afin de pouvoir évoluer dans son environnement, le robot doit être capable de prendre en compte l'évolution de l'environnement et de modéliser ces réactions dans les situations qu'il met en oeuvre. Nous définissons pour cela les événements contingents. Un événement contingent correspond en réalité à une action de l'environnement ; sa définition est donc identique à celle d'une action du robot. Nous distinguons les événements des actions car alors que le robot a un contrôle total sur le choix des actions, il ne peut en aucun cas contrôler les événements qui, une fois les préconditions vérifiées, se déclenchent automatiquement. Evidemment nous pouvons définir des événements conditionnels, non-déterministes et possibilistes.

Afin de mieux modéliser la dynamique de l'environnement, nous devons inclure une notion temporelle à notre définition des événements. Nous pouvons donc distinguer deux types d'événements :

- les événements immédiats : les effets sont instantanés dès que les préconditions sont vérifiées ($d = 0$),
- les événements retardés : les effets n'apparaissent qu'au bout d'un laps de temps $d > 0$.

Prenons l'exemple du plancher, qui, s'il est touché, déclenche instantanément l'alarme et ferme la porte :

Plancher_touché()		
Préconditions:	Contact_plancher	
Effets:	Après(0, {(Alarme,1), (Fermée(porte),1)})	

La représentation des événements que nous proposons est à la fois une extension et une restriction des événements définis pour NODEP dans [13]. En effet la représentation de NODEP ne permet pas de qualifier la possibilité d'apparition d'un événement possibiliste et ne prend pas en compte les événements conditionnels et non-déterministes ; par contre il définit un encadrement temporel des effets de l'événement (s'il apparait). Nous nous contentons ici de la durée séparant les préconditions vérifiées et les effets.

Modification due à l'introduction du temps. Notre but n'étant pas de construire un planificateur temporel, nous limitons l'utilisation du temps à l'intérêt porté aux événements. Les actions deviennent maintenant des opérations consommatrices de temps. Nous rajoutons à notre définition une durée à chaque issue d'action. Cette durée est précise (nous ne définissons pas d'intervalle d'apparition des effets : les effets sont instantanés dès que la durée est écoulée) mais conditionnelle. Si nous prenons une action A ayant deux issues possibles I_1 et I_2 dont les durées sont respectivement d_1 et d_2 ; le résultat de l'application de A à un état S à la date t apparaît soit à la date $t + d_1$ soit à la date $t + d_2$. Dans le cas des parties «plan indentique», nous ne pouvons appliquer l'action suivante qu'à $t + max(d_1, d_2)$. De plus nous proposons un encadrement temporel pour l'obtention du but. Par exemple, l'événement *Arrivée_Police* peut être modélisé par :

Arrivée_Police()		
Préconditions:	Alarme	
Effets:	¬ Fermée(porte):	Après(5, {(Police,0,9)})
	Fermée(porte):	Après(8, {(Police,0,7)})

Modification due aux événements. La mise en place des événements permet au planificateur de «contrôler» l'aspect opportuniste. En effet, bien que le robot n'ait pas la possibilité de contrôler les événements directement, il peut, par sa connaissance des événements existants et de l'environnement, déclencher sciemment un événement dont les effets seraient opportuns (voir figure 7).

Par contre la gestion des états interdits devient plus délicate. En effet, en plus d'assurer que les actions du plan ne créent pas d'états interdits, il faut aussi vérifier qu'aucun état transitoire ou non ne puisse mener par «inaction» à un état interdit. C'est pourquoi nous définissons la notion d'état sûr. Un état S est sûr si et seulement si, sans aucune action de la part du ro-

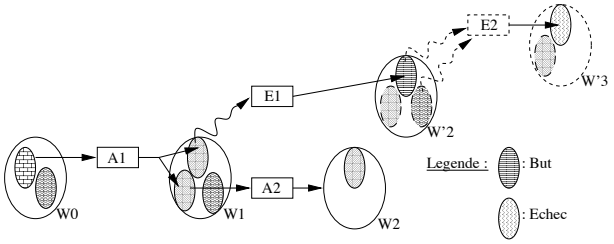

FIG. 7 – *Un exemple de plan opportuniste. Les portions d'états représentés en pointillés coorespondent aux portions qui seraient vraies si l'action A_2 n'existait pas. Nous remarquons que pour atteindre le but, le planificateur utilise l'événement E_1 (aspect opportuniste) mais est obligé de contrecarrer l'événement E_2 qui mène à un états interdit en appliquant A_2.*

bot, il est impossible d'obtenir un état interdit par le simple «jeu» des cascades successives d'événements contingents. Ce qui revient à dire que tous les événements qui se déclenchent dans l'état S ne mènent qu'à des états eux-mêmes sûrs (un état interdit n'étant évidemment pas sûr).
Pour planifier, le robot doit connaitre les états initiaux possibles ; or la planification n'est pas réalisée en un temps borné, il faut donc assurer au robot que les états initiaux qu'il a prévus sont toujours valides au début de la phase d'exécution [15]. Pour cela, nous définissons les états stables. Un état stable est un état dans lequel aucun événement ne peut s'appliquer. Notons qu'un état stable non-interdit est un état sûr.
Tous les états initiaux doivent donc être des états stables et l'état final doit être un état sûr.

3.2 Exemple

À travers l'exemple du robot cambrioleur décrit en 2.1, nous allons mettre en évidence l'intérêt des plans qualifiés et conditionnels.
Dans l'état initial, l'incertitude vient de la présence ou non d'un capteur derrière le tableau que notre robot veut dérober ; comme le montre le tableau 1, il y a souvent un capteur ($N(Capteur) = 0,7$). Rappelons que le but recherché est : $Tient(Tableau) \wedge \neg$ *Abimé(Tableau)*. Dans un premier temps, nous allons seulement demander un plan qui aboutit avec une nécessité $N_{But} \geq 0,7$. Le plan que nous obtenons (en 10ms sur une Ultra Sparc 10) est très simple :

Aller_en(L) Déconnecter(Tableau,L) Prendre(Tableau,L)

Ce plan prend le risque d'abîmer le *Tableau* dans le cas où il n'y aurait pas de capteur ($\Pi(\neg Capteur) = 0,3$). L'intérêt de qualifier le but est de permettre au planificateur de gérer lui-même les risques encourus. Ainsi, si nous demandons un plan qui n'échoue jamais ($N_{But} = 1$), le plan trouvé (212ms) comporte une action de perception sur la présence du capteur :

Aller_en(L) Tester(Tableau,L) Alarme?()
Si $Alarme \wedge Capteur$
Alors Déconnecter(Tableau,L) Prendre(Tableau,L)
Si $\neg Alarme \wedge \neg Capteur$
Alors Prendre(Tableau,L)

Nous remarquons que le plan débute par une partie «plan conformant» (incertitude sur *Capteur* puis sur *Alarme*) suivie d'une action de perception qui lève toute indétermination.
La recherche du plan se complique (5310ms) lorsque l'on introduit les événements, le plan devient alors :

Aller_en(L) Tester(Tableau,L) Alarme?()
Si $Alarme \wedge Capteur$
Alors Se_Poser() Déconnecter(Tableau,L)
Prendre(Tableau,L) Sortir()
Si $\neg Alarme \wedge \neg Capteur$
Alors Prendre(Tableau,L)

Le déclenchement de l'alarme rend maintenant possible l'arrivée de la police. Même si leur présence n'est pas certaine (nécessité de 0,9 ou 0,7 suivant les cas) et bien que dans tous les cas ce sera après l'obtention du but, l'état $Police \wedge Est\_Dans(Robot, Musée)$ doit absolument être évité. Dans cet exemple, le robot utilise de manière opportuniste l'événement *Plancher_Touché()* qui va retarder l'entrée de la police et permettre ainsi au robot de quitter le musée. En effet les durées respectives de *Alarme*?(), de *Déconnecter(Tableau,L)*, de *Prendre(Tableau,L)* et de *Sortir*() sont 1, 3, 1 et 1 ce qui est trop long (la police sera là à partir de 5). La durée de l'action *Se_Poser*() étant de 1, le robot s'en sert pour déclencher l'événement *Plancher_Touché()* pour fermer la porte et se laisser ainsi le temps de sortir.

4 L'algorithmique

L'algorithme qui est présenté ici est une extension de celui présenté dans [9]. Nous allons rappeler les bases de Graphplan dont nous nous sommes inspirés, ainsi que l'algorithme de la première version. Dans un second temps nous allons décrire l'aspect temporel de TGP [16] que nous allons utiliser pour cette nouvelle version.

4.1 Graphplan

Graphplan est un planificateur, développé par [2], qui utilise des opérateurs de type STRIPS ainsi qu'une méthode de propagation de contraintes. Partant de l'état initial comme première étape, il développe toutes les instanciations d'actions applicables de cette étape pour générer la suivante jusqu'à ce que tous les buts apparaissent (à chaque étape un fait n'apparaît qu'une

15. Toutes les dates sont définies à partir du début de l'exécution.

seule fois, les incompatibilités entre deux instances possibles sont stockées par des relations d'exclusion mutuelle appelées *mutex*). Cette première partie correspond à une propagation des *mutex*, la seconde étant la recherche du plan (recherche par arrière) en résolvant les conflits de mutex.

On se rend compte qu'une fois qu'un fait est apparu dans une étape, l'action de conservation (*No-op*) le rend présent aux suivantes ; inversement lorsqu'un *mutex* disparait, il ne pourra plus réapparaître aux étapes suivantes. Ces deux remarques permettent de compresser le graphe décrit précédemment en conservant le niveau d'apparition des faits et le niveau de destruction des *mutex*.

4.2 La version précédente de notre planificateur

Nous rappelons ici les grandes lignes de l'algorithme présenté dans [9]. L'introduction des nécessités modifie la définition des *mutex* d'«interférence» et la recherche du plan qui peut maintenant être un peu guidée par les nécessités. Notons tout de même, qu'à cause des observations, qui permettent d'augmenter la nécessité d'un fait, nous ne pouvons pas supprimer les faits dont la nécessité est inférieure à N_{But}. Pour le non-déterminisme, nous cherchons un plan correct pour un des mondes possibles et ensuite nous le validons en vérifiant s'il mène au but dans les autres cas. La recherche d'un plan incluant des observations est un peu plus longue car il faut trouver un nouveau plan par résultat d'observation (le plan se trouvant avant l'action de perception est évidemment «conformant»). La validation des événements se fait par un développement durant la phase d'expansion du graphe et un aller-retour durant la recherche du plan.

4.3 TGP

TGP est un planificateur temporel basé sur Graphplan développé par [16]. Il utilise un graphe compressé, ce qui permet d'associer directement une date d'apparition à chaque proposition et à chaque action. Le temps propagé durant la phase d'expansion du graphe (chaque action a une durée précise) ; pour cela TGP scinde les *mutex* en deux catégories : les *mutex éternels* qui sont toujours vrais (e.g. les propositions F et $\neg F$), et les *mutex conditionnels* Φ qui sont vrais lorsque Φ est satisfiable. De même chaque catégorie est divisée en trois parties : les *mutex Proposition—Proposition*, les *mutex Action—Action* et un nouveau type de *mutex*, les *mutex Action—Proposition* qui permet de mieux faire le lien entre les actions et les propositions. Une approximation récursive de Φ permet d'associer à chaque couple (X, Y) le *mutex conditionnel* $\lfloor X < t_X \wedge Y < t_Y$. $\lfloor X$ correspond au début de l'exécution de X si X est une action ou à la date à laquelle X devient vraie dans le plan si X est une proposition. t_X est calculé en fonction des créations possibles de X (i.e. ses préconditions si X est une action ou les actions ayant X pour effet sinon) et de son utilisation. Par exemple le *mutex conditionnel* de l'action A avec l'action B se calcule à partir d'une approximation de $\Phi = \bigvee_i \Phi_i \vee \bigvee_j \Psi_j$ avec Φ_i *mutex* de A et P_i (P_i préconditions de B) et Ψ_j *mutex* de B et Q_j (Q_j préconditions de A). La recherche par l'arrière du plan, qui est guidée par les *mutex* créés, prend explicitement en compte la date à laquelle chaque but ou sous-but doit être vrai.

4.4 L'algorithme utilisé

Nous utilisons le graphe condensé et la gestion des *mutex* mis en place dans TGP pour modéliser le temps. Durant la phase d'expansion du graphe, nous développons indifféremment les actions et les événements (toutes issues confondues). Le fait qu'un événement applicable soit toujours déclenché ne modifie que la recherche du plan. En effet, pour chaque état créé, le planificateur doit vérifier que tous les états suivants sont sûrs (pas d'influence sur l'expansion du graphe) ; l'«opportunisme» face aux événements est permis grâce l'insertion des événements aux côtés des actions dans les deux phases (développement du graphe et recherche du plan).

Le tableau 2 décrit de manière synthétique l'algorithme utilisé. Nous utilisons trois structures différentes : $\mathcal{B}$,$\mathcal{P}$ et $\mathcal{E}_{init}$. $\mathcal{B}$ correspond à la liste des buts (triplés (Fait, nécessité minimale, date au plus tard)) à réaliser. Cette liste est triée par ordre temporel décroissant. $\mathcal{P}$ est le plan, il est modélisé sous la forme d'un arbre à cause des branches créées par les actions de perception (chaque action est associée à une date de début). $\mathcal{E}_{init}$ correspond à l'ensemble des états initiaux possibles.

5 Conclusion

Dans cet article, nous avons présenté une nouvelle modélisation de l'action et du changement qui permet de prendre en compte la dynamique de l'environnement dans lequel le robot évolue. Nous avons, dans un premier temps, défini deux sources de non-déterminisme dans l'issue d'une action, l'une due à une modélisation insuffisante, non-déterminisme de modélisation, et l'autre due à un manque d'informations sur l'état du monde, non-déterminisme d'incertitude. Dans un second temps, nous avons défini les événements contingents qui permettent de prévoir correctement (dans la limite du modèle) les effets de bord de certaines actions ou inactions.

Chercher_Plan($\mathcal{B}$,$\mathcal{P}$,$\mathcal{E}_{init}$)
Dépiler (B, N_B, t_B) de $\mathcal{B}$
Si $t = 0$ **Alors**
Si $B \in \mathcal{E}_{init}$ **Alors**
Retourner Chercher_Plan($\mathcal{B}$,$\mathcal{P}$,$\mathcal{E}_{init}$)
Sinon Retourner *Echec*
Sinon
Soit S l'ensemble des actions et événements ayant une issue contenant B, $N(B|P) \geq N_B$ et dont la date de fin au plus tôt t vérifie $t \leq t_B$.
Soit *Retour* $\leftarrow$ *Echec*
Tant que *Retour* = *Echec* **faire**
Choisir $X \in S$ compatible avec $\mathcal{P}$ (non-mutex) et minimisant la nécessité
Si X n'est pas une action de perception **Alors**
Pour chaque issue de X **faire**
Si $\mathcal{P}$ ne s'applique pas ou s'il créé un état non-sûr **Alors** *Retour* $\leftarrow$ *Echec*; *break*
Sinon
Si $\mathcal{P}$ ne s'applique pas à l'issue contenant B ou s'il créé un état non-sûr **Alors** *Retour* $\leftarrow$ *Echec*; *break*
Soit $\mathcal{B}' \leftarrow \mathcal{B}$
Insérer les préconditions de X dans $\mathcal{B}$
Ajouter X à $\mathcal{P}$
Soit *Fin* $\leftarrow \mathcal{P}$
Retour $\leftarrow$ Chercher_Plan($\mathcal{B}$,$\mathcal{P}$,$\mathcal{E}_{init}$)
Si *Retour* = *OK* **Alors**
Si X est une action de perception **Alors**
Soit *Début* $\leftarrow \mathcal{P}$−*Fin*
Pour chaque issue de X ne contenant pas B **faire**
Créer l'état $\mathcal{E}_{courant}$ à partir de Début
Soit $\mathcal{P}' \leftarrow \phi$
Retour $\leftarrow$ *Retour* et Chercher_Plan($\mathcal{B}'$,$\mathcal{P}'$,$\mathcal{E}_{courant}$)
Si *Retour* = *OK* **Alors**
Ajouter chaque branche $\mathcal{P}'$ à $\mathcal{P}$
Retourner *Retour*

TAB. 2 – *Algorithme de recherche de plan par l'arrière.*

Nous proposons ensuite un algorithme pour la recherche de plan. Le plan ainsi obtenu, qui peut être conditionnel grâce aux actions de perception qui lèvent certaines indéterminations, est qualifié en termes de nécessité de réussite ce qui permet au planificateur de ne développer que les branches nécessaires. Nous garantissons tout de même sa robustesse par l'introduction d'états stables et d'états sûrs.

La définition des états stables pourrait être étendue aux situations, graphes d'états dont les transitions seraient seulement des actions reflexes et des événements contingents. Une telle extension permettrait d'élargir la description des situations initiales possibles aux comportements réactifs du robot. De plus, la mise en place de priorités entre les buts permettrait d'assouplir la notion d'états interdits et ainsi de favoriser dans certains cas la prise de risque par le robot.

Références

[1] C. Anderson, D. Smith, and D. Weld. Conditional effects in graphplan. In *Proc. 4th Int. Conf. AI Planning Systems*, 1998.

[2] A.L. Blum and M.L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, pages 281–300, 1997.

[3] A.L. Blum and J.C. Langford. Probabilistic planning in the graphplan framework. *AIPS Workshop on "Planning as Combinatorial Search"*, 1998.

[4] C. Da Costa Pereira. Planification d'actions en environnement incertain: une approche fondee sur la theorie des possibilites. In *PhD thesis, Universite Paul Sabatier, Toulouse*, 1998.

[5] D. Draper, S. Hanks, and D. Weld. Probabilistic planning with information gathering and contingent execution. In *Proc. 2nd Int. Conf. AI Planning Systems*, pages 31–36, 1994.

[6] D. Dubois and H. Prade. Possibility theory - an approach to computerized processing of uncertainty. *Plenum Press*, 1988.

[7] D. Dubois and H. Prade. Possibility theory as a basis for qualitative decision theory. *Proc. 14th Inter. Joint Conf. on Artificial Intelligence (IJCAI'95)*, pages 19–25, 1995.

[8] R.E. Fikes and N.L. Nilsson. Strips: A new approach to the application of therom proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[9] E. Guéré and R. Alami. A possibilistic planner that deals with non-determinism and contingency. *Proc. 16th Inter. Joint Conf. on Artificial Intelligence (IJCAI'99)*, 1999.

[10] J. Koehler, B. Nebel, J. Hoffmann, and Y. Dimopoulos. Extending planning graphs to an adl subset. In *4th European Conference on Planning (ECP'97)*, 1997.

[11] N. Kushmeric, S. Hanks, and D. Weld. An algorithm for probabilistic planning. *Artificial Intelligence*, 2:239–286, 1995.

[12] B. Nebel, Y. Dimopoulos, and J. Koehler. Ignoring irrelevant facts and operators in plan generation. In *4th European Conference on Planning (ECP'97)*, 1997.

[13] J. Perret. Teleprogrammation au niveau tache pour un robot mobile autonome d'intervention sur site distant. In *PhD thesis, Universite Paul Sabatier, Toulouse*, 1995.

[14] L. Pryor and G. Collins. Planning for contingencies: A decision-based approach. *Artificial Intelligence Research*, 1996.

[15] D. Smith and D. Weld. Conformant graphplan. In *Proc. 15th Nat. Conf. AI.(AAAI'98)*, 1998.

[16] D. Smith and D. Weld. Temporal planning with mutual exclusion reasoning. *Proc. 16th Inter. Joint Conf. on Artificial Intelligence (IJCAI'99)*, 1999.

[17] D. Weld, C. Anderson, and D. Smith. Extending graphplan to handle uncertainty and sensing actions. In *Proc. 15th Nat. Conf. AI.(AAAI'98)*, 1998.

[18] L.A. Zadeh. Fuzzy sets as a basis for theory of possibility. *Fuzzy Sets and Systems*, 1:3–28, 1978.