
Hiérarchisation dynamique de la recherche : Application au planificateur IXTET

Frédéric Garcia †

Unité de Biométrie et Intelligence Artificielle, INRA

BP 27, 31326 Castanet Tolosan cedex, FRANCE

Frederic.Garcia@toulouse.inra.fr - www-bia.inra.fr/T/garcia

Philippe Laborie ‡

LAAS/CNRS, 7 avenue du Colonel Roche, 31077 Toulouse cedex, FRANCE

Philippe.Laborie@irisa.fr

RÉSUMÉ. Nous décrivons dans cet article la génération automatique et l'utilisation de hiérarchies d'abstraction pour le planificateur IXTET. Deux idées sont développées. Dans un premier temps nous étendons le formalisme d'abstraction à la représentation d'IXTET qui permet l'utilisation d'attributs valués, de ressources et de contraintes temporelles. Ensuite, nous proposons une nouvelle manière de gérer dynamiquement la hiérarchie d'abstraction en cours de planification. Cette approche de moindre engagement pour la hiérarchie a été implémentée sur le système de planification IXTET, où son intérêt en terme de réduction du nombre de retour-arrières et d'accroissement de l'efficacité globale a été clairement démontré sur plusieurs domaines de planification réalistes.

ABSTRACT. Hierarchical problem solving with abstraction is a widely adopted strategy to explore very large search trees. This paper describes the automatic generation of abstraction hierarchies for the IXTET planner. Two points are mainly developed. First, we extend the classical STRIPS-like abstraction formalism to the IXTET representation. Second, we describe a new way of dynamically managing abstraction hierarchies during planning that follows a least-commitment strategy. This hierarchy has been implemented on the IXTET planner and its interest has been clearly established on several realistic domains.

MOTS CLÉS : *planification, hiérarchie d'abstraction.*

KEYWORDS : *planning, abstraction hierarchy.*

† Les travaux ayant conduit à cet article remontent au séjour post-doctoral de Frédéric Garcia au LAAS, entre avril et août 1994, financé par le PRC-IA.

‡ Philippe Laborie travaille actuellement à l'IRISA, Rennes.

1 Introduction

L'utilisation de techniques de hiérarchisation se révèle en général très fructueuse lorsqu'il s'agit de résoudre des problèmes de planification de grande taille. De manière générale, la hiérarchisation de la recherche consiste à d'abord résoudre

le problème à un niveau d'abstraction élevé puis à affiner progressivement la solution dans des niveaux moins abstraits jusqu'à aboutir, finalement, à une solution concrète au problème.

Dans les approches hiérarchiques classiques de la planification [SAC 74] [YAN 90] [KNO 94], cette décomposition hiérarchique en niveau doit être complètement définie hors-ligne, soit par l'utilisateur, soit automatiquement à partir d'une analyse syntaxique des opérateurs de planification. Au cours du processus de planification, le contrôle de la recherche n'a alors plus aucun moyen d'action sur cette hiérarchie (notion de *hiérarchie statique*). L'originalité de l'approche hiérarchique que nous présentons ici tient en particulier dans le fait que nous ne nous engageons pas à une hiérarchie complètement spécifiée hors-ligne et imposée au contrôle. A un domaine donné, nous n'associons pas une seule hiérarchie mais un jeu de contraintes permettant de caractériser un *ensemble de hiérarchies admissibles* vis-à-vis d'un critère donné. Le choix final de la hiérarchie effectivement utilisée est alors contrôlé par le processus de planification lui-même (notion de *hiérarchie dynamique*).

D'autre part la hiérarchisation utilisée dans IXTE, du fait de l'intégration entre synthèse de plans et gestion de ressources, permet d'englober et d'entrelacer de manière naturelle ces deux aspects tout au long de la chaîne allant du niveau le plus abstrait au niveau le plus concret.

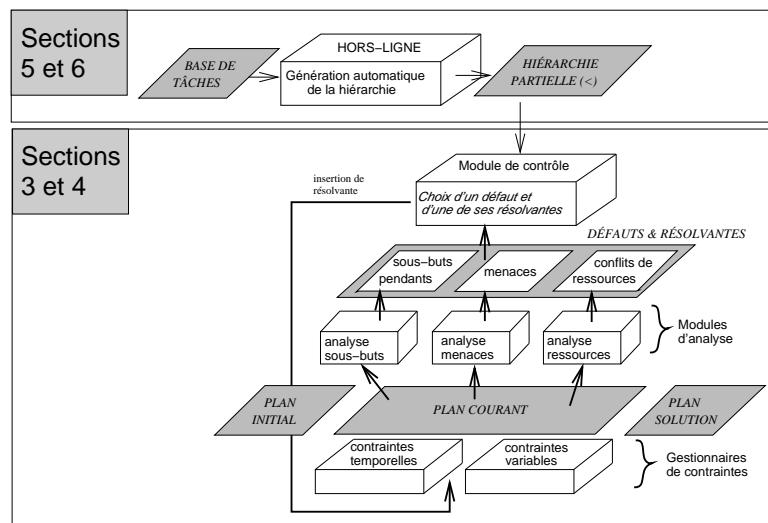


Figure 1. Architecture du système hiérarchisé

Cet article décrit notre approche, ses extensions possibles et un certain nombre de résultats empiriques montrant son intérêt. La première section résume l'état de l'art des recherches portant sur la hiérarchisation par abstraction appliquée à la planification. Les deux sections suivantes (sections 3 et 4) décrivent le système de planification IXTE, son formalisme et l'algorithme général de recherche d'un plan (voir figure 1).

L'approche hiérarchique que nous avons développée pour IXTE_T est présentée dans sa généralité en section 5, et des développements potentiels sont suggérés par la suite (section 6). Différents résultats expérimentaux sont alors donnés en section 7.

2 Hiérarchisation par abstraction en planification

La plupart des algorithmes actuels de planification reposent sur le principe d'exploration d'un arbre de recherche dont les sommets correspondent à des plans partiels en cours de construction, et les arêtes à l'addition d'opérateurs et de contraintes en vue d'établir des sous-buts ou de façon plus générale de résoudre des défauts présents dans ces plans partiels. Le problème de la sélection du sous-but à établir ou du prochain défaut du plan partiel à résoudre est alors une composante majeure de la stratégie de l'algorithme de contrôle qui doit être définie pour chaque système de planification.

Récemment, de nombreux travaux ont été effectués visant à formaliser et gérer des *hiérarchies d'abstraction* en planification linéaire aussi bien que non-linéaire, avec pour objectif de définir une procédure intelligente de sélection des défauts [YAN 90, KNO 91b, KNO 94]. Plusieurs résultats expérimentaux ont montré le fait que l'utilisation d'une hiérarchie d'abstraction pouvait permettre d'augmenter de manière substantielle les performances de la planification [WIL 88, KNO 94].

Le principe des hiérarchies d'abstraction utilisées en planification, initialement dû à Sacerdoti dans le cadre du planificateur ABSTRIPS [SAC 74] et réintroduit plus tard dans des systèmes comme ABTWEAK [YAN 90] ou PRODIGY [KNO 94] est très simple. Un entier naturel, souvent appelé *niveau d'abstraction* ou *criticité* est associé à chaque motif de sous-but potentiel du domaine de planification considéré. Généralement, en partant du niveau de criticité le plus élevé, le problème de planification est résolu à chaque niveau i en ne considérant à ce niveau là que les sous-but ayant une criticité supérieure ou égale à i . Cette approche fait apparaître deux types de problèmes : (1) comment associer des niveaux d'abstraction à chaque sous-but potentiel et (2) comment gérer ces niveaux dans la procédure de recherche d'un plan solution.

Tous les résultats sur les hiérarchies d'abstraction ont jusqu'ici été obtenus dans le cadre d'une représentation de type STRIPS. Dans ce formalisme, une action est représentée par un opérateur α comme une paire $(pre(\alpha), effect(\alpha))$ où $pre(\alpha)$ et $effect(\alpha)$ sont des listes de littéraux, c'est-à-dire des formules atomiques (positives ou négatives) de la logique du premier ordre décrivant respectivement les préconditions et les effets de chaque action. Les sous-but sont alors représentés par des littéraux auxquels, dans le contexte d'une hiérarchie d'abstraction, seront associés des niveaux.

Étant donné un domaine de planification et un problème à résoudre, plusieurs hiérarchies d'abstraction peuvent être envisagées. Afin de caractériser ces hiérarchies, divers critères peuvent être définis. Les deux critères les plus étudiés sont respectivement la propriété d'*affinement descendant* (*Downward Refinement Property* ou *DRP*) [BAC 91, BAC 94] et la propriété de *monotonie ordonnée* (*Ordered Monotonicity Property* ou *OMP*) [KNO 90] décrites informellement ci-dessous.

Dans ces définitions, un plan abstrait est un plan solution du problème de planification à un niveau d'abstraction i donné. Un plan P_{i-1} est un *affinement* d'un

plan abstrait P_i si P_{i-1} est un plan abstrait qui ne diffère de P_i que par l'addition d'opérateurs et/ou de contraintes ayant été insérées pour satisfaire des préconditions introduites au niveau $i - 1$.

Définition 1

affinement descendant (DRP) : *S'il existe une solution au problème global, chaque plan abstrait peut être raffiné en une solution.*

monotonie ordonnée (OMP) : *Pour chaque plan abstrait, aucun affinement de ce plan n'affectera les littéraux déjà établis dans celui-ci.*

Il est clair que la DRP est la propriété la plus intéressante qu'une hiérarchie puisse satisfaire, pouvant conduire sous certaines conditions idéales à une réduction exponentielle du coût de la recherche [KNO 91a]. Néanmoins, il s'agit d'un critère très fort et donc, qui ne peut être garanti que dans des cas très particuliers. L'OMP est plus réaliste et donc, plus intéressante pour des problèmes généraux qui se posent en pratique. Lorsque cette propriété est vérifiée, beaucoup de sources de retour-arrière sont éliminées puisqu'aucune interaction directe entre les niveaux d'abstraction n'est possible. Dans ce cas, l'efficacité de la recherche peut être grandement améliorée. De plus, il a été montré que des hiérarchies d'abstraction vérifiant l'OMP pouvaient facilement être générées automatiquement, ce qui rend cette propriété d'autant plus intéressante [KNO 94].

Pour construire automatiquement une hiérarchie d'abstraction, Knoblock a proposé différentes conditions suffisantes permettant de garantir l'OMP. Ces conditions correspondent à un ensemble de contraintes sur le niveau des littéraux du domaine ; contraintes qui peuvent être directement extraites de la description syntaxique des opérateurs :

Propriété 1 (conditions suffisantes pour l'OMP)

Soit O l'ensemble des opérateurs du domaine. $\forall \alpha \in O$,
 $\forall p \in pre(\alpha)$ et $\forall q, q' \in effect(\alpha)$,

1. $niveau(q') = niveau(q)$
2. $niveau(p) \leq niveau(q)$

Le principe de ces conditions est simple : elles imposent que chaque littéral qui pourrait être modifié par le processus de résolution d'un autre littéral ait un niveau d'abstraction inférieur ou égal à ce dernier. Ceci assure que tout opérateur inséré pour satisfaire, directement ou indirectement un littéral, laissera invariants les littéraux de plus haut niveau d'abstraction.

A partir de l'ensemble des conditions suffisantes obtenues en analysant le domaine, il est aisément de générer une hiérarchie d'abstraction vérifiant l'OMP. Pour cela, il suffit de trouver une valuation des littéraux qui satisfasse l'ensemble des contraintes. La solution proposée dans ALPINE [KNO 94] consiste à construire un graphe orienté

dont les nœuds représentent des littéraux (ou des ensembles de littéraux de même niveau) et les arcs, les contraintes de niveau hiérarchique entre les littéraux représentés dans les nœuds. Un tri topologique du graphe est alors effectué qui définit un ordre total sur les littéraux ; ordre total conduisant à une hiérarchie d'abstraction qui vérifie nécessairement l'OMP. Le choix de l'ordre total construit à partir du graphe est crucial relativement à l'efficacité de la planification. Dans [BÄC 95], les auteurs ont montré qu'un mauvais choix pouvait conduire, même pour des hiérarchies vérifiant la DRP, non pas à une amélioration des performances mais bien à une détérioration exponentielle de celles-ci ; nous verrons plus loin comment l'approche que nous proposons aide à résoudre ce problème en ne s'engageant pas sur le choix d'un ordre total [GAR 96].

Quelques extensions à ce modèle ont été proposées [KNO 94, FIN 93]. Fink et Yang ont introduit la notion de *précondition interdite*, pour une précondition l d'une action α telle qu'une fois que l devient fausse, les préconditions de α ne pourront plus jamais être établies. Restreindre les conditions suffisantes précédentes aux seuls préconditions *non-interdites* définit alors des conditions nécessaires et suffisantes pour l'OMP. Toutefois, il semble difficile de définir un algorithme efficace permettant de détecter les préconditions interdites, ce qui limite l'intérêt pratique de ce résultat.

Il peut être avantageux de prendre aussi en compte le problème à résoudre lors de la génération de la hiérarchie d'abstraction. L'intérêt étant que les hiérarchies générées seront moins sur-contraintes puisque les contraintes entre niveaux d'abstraction qui ne sont pas pertinentes vis-à-vis du problème ne seront pas prises en compte. La solution adoptée par Knoblock consiste à ne considérer comme effets possibles d'une action, lors de la génération automatique des contraintes sur les niveaux, que ceux qui sont *pertinents* pour la résolution des buts du problème.

Une autre extension possible revient à faire la distinction entre *effets primaires* et *effets secondaires* d'un opérateur. Classiquement, les effets primaires spécifient la justification réelle de la présence d'une action dans un plan tandis que les effets secondaires sont des effets de bord de l'action. Durant le processus de planification, les opérateurs sont sélectionnés pour satisfaire un sous-but uniquement sur la base de leurs effets primaires ; ceci permet une nette amélioration des performances suite à une réduction du facteur de branchement dans l'arbre de recherche [FIN 95]. Dans les conditions suffisantes pour vérifier l'OMP, seuls les effets primaires nécessitent alors d'être pris en compte. Ici encore, l'intérêt est d'obtenir un réseau moins contraignant sur les niveaux d'abstraction.

La dernière extension est liée à l'utilisation de schémas d'opérateurs. En effet, la majorité des systèmes de planification utilise des schémas d'opérateurs plutôt que des opérateurs totalement instanciés. La solution proposée dans ALPINE est d'associer un *type* à chacune des constantes pouvant apparaître dans les arguments d'un littéral et de construire, à partir de cette typologie des constantes (ou objets) du domaine, une typologie des littéraux instanciés. Les niveaux d'abstraction sont alors associés aux différents types de littéraux plutôt qu'aux littéraux eux-mêmes.

3 Le système de planification IXTE

Le système de génération de plan d’actions basé sur le formalisme IXTE est développé au LAAS/CNRS depuis une dizaine d’années sous la direction de Malik Ghallab. L’un des objectifs principaux du système IXTE est de pouvoir résoudre une large gamme de problèmes réalistes tout en assurant certaines propriétés de l’algorithme de recherche (en particulier sa complétude). Le lecteur désirant plus de détails sur le planificateur IXTE pourra se reporter à [LAR 94, LAB 95a].

Ce planificateur se distingue tout particulièrement des planificateurs dits classiques par la prise en compte explicite du temps en ce qui concerne la description des objectifs à atteindre et des opérateurs de planification, et par la représentation de l’état de l’environnement sous forme d’attributs d’état valués et d’attributs de ressources. Une distinction claire entre ces différents éléments (contraintes temporelles, attributs d’états et de ressources) permet alors de définir des algorithmes spécifiques à ces domaines, qui sont intégrés au sein de l’architecture globale du système de planification.

3.1 Représentation du monde

A un instant donné, l’état courant du monde est décrit par un ensemble d’*attributs d’état valués* et un ensemble d’*attributs de ressources*. Notre représentation du monde est supposée **complète et exhaustive** : toutes les connaissances relatives à l’état du monde à un instant donné sont entièrement explicitées. Nous supposons d’autre part que nos opérateurs de planification (appelés *tâches*) sont **déterministes**.

3.1.1 Les attributs d’état

Un **attribut d’état** permet de décrire une caractéristique non-cumulative du monde à un instant donné ; par exemple, la localisation d’un robot `robot1` dans une zone donnée `zone1` est représentée par l’attribut `localisation(robot1):zone1`. Par hypothèse, un attribut d’état ne peut prendre qu’une seule valeur à un instant donné.

Un des intérêts principaux de l’utilisation d’attributs valués plutôt que de relations binaires est qu’ils permettent d’exprimer de manière tout à fait naturelle des contraintes telles que la non-ubiquité sans avoir pour cela besoin de définir de règle particulière.

Les domaines possibles pour les arguments d’un attribut d’état sont explicitement déclarés. Ainsi dans le cas ci-dessus on aura :

```
attribute localisation(?robot){  
    ?robot in {robot1, robot2};  
    ?value in {zone1, zone2, zone3};  
}
```

Selon leur dynamicité, il est possible de distinguer des types d’attributs d’états **rigides** (pas de changement de valeur au cours du temps), **flexibles** et, parmi ces derniers, des attributs **contrôlables** (tous les changements de valeur sont contrôlés par l’agent qui planifie) ou **contingents** (non-contrôlables). Notre approche permet de gérer de manière homogène et indifférenciée l’ensemble de cette typologie.

3.1.2 Les attributs de ressource

Une **ressource** est définie comme toute substance ou ensemble d'objets dont le coût ou la quantité disponible induit des contraintes sur les actions qui l'utilisent.

La différence essentielle entre un attribut de ressource et un attribut d'état est que l'état est modifié de manière *absolue* par l'action tandis que les ressources ne sont affectées que de manière *relative* en augmentant ou diminuant la quantité disponible. S'il est possible de représenter l'utilisation de ressources non-partageables par un attribut d'état (du type `ressource.libre` et prenant une valeur booléenne), l'utilisation de *ressources partageables* ne peut pas être représentée sous la forme d'événement sur des attributs d'état.

Pour une action donnée, plusieurs ressources peuvent être considérées comme équivalentes. Deux ressources sont dites de même *type* si et seulement si il existe au moins une action qui peut utiliser indifféremment l'une ou l'autre de ces ressources. Les ressources d'un type donné sont explicitement déclarées ; par exemple :

```
resource puissance_electrique(?batterie){  
    ?batterie in {batteriel, batterie2};  
    capacity(batteriel)=50;  
    capacity(batterie2)=60;  
}
```

Un **attribut de ressource** permet de spécifier une certaine quantité de ressource utilisée par une action à un instant donné ; par exemple `puissance_electrique(batteriel):10`.

3.2 Représentation des contraintes

Pour des raisons de complexité algorithmique, le **gestionnaire de contraintes temporelles** d'IXTET repose sur une représentation à base d'**instants**. Sont gérées à la fois :

- des contraintes temporelles symboliques (i.e. contraintes de précédence : $t < t'$, de simultanéité : $t = t'$) ; et
- des contraintes numériques exprimées sous la forme d'un intervalle de réels bornant la distance temporelle entre deux instants (du type $t - t' \text{ in } [d_{min}, d_{max}]$).

Les disjonctions de contraintes temporelles ne sont pas directement gérées à ce niveau mais au niveau de l'algorithmique de contrôle.

Toutes les variables atemporelles utilisées dans les attributs sont considérées comme ayant un domaine de valeurs possibles discret et fini. Le **gestionnaire de contraintes sur les variables** permet d'exprimer les contraintes suivantes :

- appartenance à un domaine : $?x \text{ in } \{X_1, X_2, \dots, X_n\}$
- égalité : $?x = ?x'$

- inégalité: $?x \neq ?x'$
- dépendance: $?x \text{ in } \{X_1, X_2, \dots, X_k\} \Rightarrow ?y \text{ in } \{Y_1, Y_2, \dots, Y_l\}$

Le rôle du gestionnaire de contraintes sur les variables est de vérifier la cohérence globale du réseau, de propager ces contraintes et de répondre à des requêtes du type : deux variables peuvent-elles s'unifier ? Peuvent-elles prendre simultanément des valeurs différentes ? Les contraintes d'inégalité ou de dépendance entre variables sont propagées par des techniques classiques de consistance d'arc¹.

3.3 Représentation de la persistance et du changement

IXTET repose sur une logique réifiée où les attributs d'état et de ressources sont qualifiés temporellement par les prédictats *event* et *hold* d'une part, et par les prédictats *use*, *consume* et *produce* d'autre part.

- une assertion $hold(att(x_1, \dots, x_n) : v, (t_1, t_2))$ exprime la persistance de la valeur de l'attribut $att(x_1, \dots, x_n)$ à v sur l'intervalle temporel $[t_1, t_2]$. A noter que t_1 et t_2 sont des instants (variables temporelles) du réseau de contraintes temporelles et x_1, \dots, x_n des variables atemporelles qui peuvent supporter des contraintes (cf. § 3.2).
- un événement $event(att(x_1, \dots, x_n) : (v_1, v_2), t)$ représente un changement instantané de valeur à l'instant t pour l'attribut $att(x_1, \dots, x_n)$ de l'ancienne valeur v_1 à la nouvelle valeur v_2 . Les événements permettent d'exprimer des changements discrets du monde ; nous ne représentons pas les changements continus comme cela est fait dans [PEN 94].

Nous distinguons trois modes d'utilisations possibles d'une ressource par une action :

- la *consommation* d'une certaine quantité q de ressource à un instant donné t : après exécution de l'action la ressource devient moins disponible qu'elle ne l'était avant: $consume(res(x_1, \dots, x_n) : q, t)$ (ex: consommation de carburant) ;
- la *production* de ressource à un instant donné: $produce(res(x_1, \dots, x_n) : q, t)$ (ex : production de carburant par l'action “faire le plein”);
- l'*emprunt* de ressource pour lequel la capacité de la ressource n'est affectée que sur un intervalle temporel donné: $use(res(x_1, \dots, x_n) : q, (t_1, t_2))$ (ex : utilisation d'un outil pour une action).

Nous faisons l'hypothèse que les quantités q utilisées sont des constantes de \mathbb{R} .

1. Pour des raisons de complexité nous nous contentons dans ce cas d'une cohérence locale qui ne remet toutefois pas en cause la complétude du système de planification.

3.4 Le formalisme de tâche

Dans IXET, les schémas d'opérateurs de planification sont appelés **tâches**. Une hiérarchie de tâches est définie où chaque tâche est composée :

- d'un ensemble de sous-tâches ;
- d'un ensemble d'événements permettant de décrire les changements du monde causés par l'exécution de la tâche ;
- d'un ensemble d'assertions sur des attributs d'état permettant d'exprimer certaines conditions à l'exécution de la tâche ainsi que la protection de certains faits nécessaires à son bon déroulement ;
- d'un ensemble de propositions d'utilisation de ressources spécifiant quelles ressources sont utilisées par la tâche, en quelle quantité et de quelle manière ;
- d'un ensemble de contraintes temporelles et de contraintes sur les variables liant les différentes variables (temporelles ou non) de la tâche.

Les sous-tâches auxquelles fait référence une tâche donnée sont spécifiées avec la même syntaxe. Cette représentation hiérarchique ne doit pas être récursive ; de plus il s'agit uniquement d'une hiérarchie de représentation visant à simplifier l'écriture des tâches par l'opérateur. Cette hiérarchie n'est pas, à l'heure actuelle, liée à la hiérarchie de contrôle qui fait l'objet de cet article.

Des exemples de tâches élémentaires (ne contenant pas de sous-tâche) sont donnés en annexe dans le cadre de la planification d'expériences de biologie embarquées dans une station spatiale. Ainsi la tâche ALLER_A consiste pour un robot ?r à se déplacer depuis une zone ?Zone1 jusqu'à une zone ?Zone2. Cette tâche a bien entendu des effets et des conditions sur la localisation du robot (événements sur l'attribut LOCALISATION) mais spécifie aussi l'utilisation de certaines ressources pendant sa durée d'exécution (en l'occurrence ici, le robot lui-même qui devient indisponible pour effectuer d'autre tâches, la puissance électrique consommée par le robot, l'utilisation d'une partie du canal de communication et d'une zone de déplacement). La durée de la tâche s'étalera entre 2 et 4 mn.

Le problème initial \mathcal{P}_{init} est une tâche particulière qui décrit le scénario du problème, c'est-à-dire :

- la valeur initiale pour l'ensemble des instances des attributs d'état ;
- les changements attendus au niveau de certains attributs d'état contingents qui ne seront pas contrôlés par le système ;
- les quantités de ressources disponibles et l'évolution temporelle de la quantité disponible de ressources ;
- les buts qui doivent être résolus.

IXTET permet une flexibilité totale quant à l'expression de contraintes temporelles entre ces différents éléments. Voici un exemple très simplifié de problème pour le monde décrit ci-dessus :

```
task PB_COLUMBUS_SIMPLE() (debut,fin){
    // situation initiale
    explained event(POSITION(Culture1):(?,Stock),debut);
    explained event(INCUBE(Culture1,T30):(?,false),debut); //...
    // buts
    hold(INCUBE(Culture1,T30):true, (t_but,fin)); //...
    debut <= t_but <= fin;
    (fin - debut) in [00:00:00, 00:30:00];}
```

Initialement, la culture Culture1 est dans la zone Stock et non incubée. L'objectif est qu'elle soit incubée d'ici une demi-heure².

4 La synthèse d'un plan d'action

Comme tous les planificateurs basés sur une recherche dans un espace de plan partiels [BAR 92, PEN 92], notre système de planification explore un arbre de recherche dont la racine est le problème initial, les nœuds sont des plans partiels et les feuilles des plans solution. Nous présentons ici l'algorithme général de recherche d'une solution.

4.1 Critère de plan solution et algorithme général

A un nœud donné de l'arbre de recherche correspond un plan partiel courant \mathcal{P} . Sur ce plan partiel nous distinguons 3 types de **défauts** qui vont nécessiter un affinement ultérieur de ce plan partiel : les propositions non-expliquées (ou sous-buts), les contraintes sur des persistances risquant de devenir invalides (menaces sur des protections) et les conflits de ressources potentiels.

Définition 2 (proposition expliquée)

Dans un plan partiel \mathcal{P} , une proposition temporelle $hold(att(x_1, \dots, x_n) : v, (t, t^))$ ou $event(att(x_1, \dots, x_n) : (v, v^*), t)$ sera dite expliquée si et seulement si il existe un événement établisseur $event(att(x'_1, \dots, x'_n) : (v_*, v'), t')$ tel que :*

1. nécessairement $(t' \leq t)$, $(v' = v)$, $(x'_1 = x_1)$, ... et $(x'_n = x_n)$; et
2. si $(t' < t)$, il existe une assertion (appelée lien causal) $hold(att(x_1, \dots, x_n) : v, (t', t))$ entre l'événement établisseur et la proposition.

Une menace est une proposition temporelle (événement ou assertion) qui risque de violer une persistance imposée (assertion). Plus précisément :

2. Le terme `explained` est utilisé devant une proposition temporelle afin de signifier au planificateur qu'elle est vérifiée dans l'état initial et qu'elle n'a pas à être planifiée.

Définition 3 (menace sur une protection)

Une menace est :

1. soit un couple $< e, cl >$ où $e = \text{event}(\text{att}(x'_1, \dots, x'_n) : (v_*, v^*), t')$ et $cl = \text{hold}(\text{att}(x_1, \dots, x_n) : v, (t_1, t_2))$ tel qu'aucune des contraintes suivantes ne puisse être déduites du plan partiel \mathcal{P} : $\{(t' < t_1); (t' = t_1 \text{ et } v = v^*); (t_2 < t'); (t' = t_2 \text{ et } v = v_*); (x_1 \neq x'_1); \dots; (x_n \neq x'_n)\}$; ou
2. soit un couple $< h, cl >$ où $h = \text{hold}(\text{att}(x'_1, \dots, x'_n) : v', (t_*, t^*))$ et $cl = \text{hold}(\text{att}(x_1, \dots, x_n) : v, (t_1, t_2))$ tel qu'aucune des contraintes suivantes ne puisse être déduites du plan partiel \mathcal{P} : $\{(t^* < t_1); (t_2 < t_*); (x_1 \neq x'_1); \dots; (x_n \neq x'_n); (x_1 = x'_1 \text{ et } \dots \text{ et } x_n = x'_n \text{ et } v = v')\}$

Un conflit de ressource est un ensemble de propositions d'utilisations de ressources qui risque de causer une surconsommation de la ressource étant donné sa capacité maximale [LAB 95b].

Définition 4 (conflit de ressources)

Un conflit de ressources est un ensemble $\{u_1, u_2, \dots, u_k\}$ de propositions d'utilisations de ressources d'un même type res tel que (si $u_i = \text{use}(res(x_{i,1}, \dots, x_{i,n}) : q_i, (t_i^-, t_i^+))$) :

1. aucune des contraintes $\{(t_i^+ < t_j^-)_{(i \neq j)}\}$ ne peut être déduite de \mathcal{P} (i.e., les propositions peuvent s'intersecter globalement dans le temps) ;
2. il existe un n -uplet de constantes (X_1, \dots, X_n) tel que la conjonction de contraintes sur les variables $(x_{i,j} = X_j)$ pour $(i, j) \in [1, k] \times [1, n]$ n'est pas incohérente avec \mathcal{P} ; et
3. $\sum_{i=1}^k q_i > Q(res(X_1, \dots, X_n))$ où Q désigne la capacité maximale des ressources.

On définit alors la notion de plan solution :

Définition 5 (plan solution)

Un plan partiel \mathcal{P} sera dit **plan solution** si et seulement si :

1. le réseau des contraintes temporelles est consistant ;
2. le réseau des contraintes sur les variables atemporelles est consistant ; et
3. il ne contient plus aucun défaut.

Au cours de la recherche, un arbre de plan partiel est parcouru. Un module de contrôle déroule l'algorithme général non-déterministe de la figure 2 en faisant appel à chaque étape à trois modules d'analyse du plan, chacun étant chargé d'analyser un type particulier de défaut présent dans le plan courant.

Algorithme planifier(\mathcal{P})
1. terminaison : si \mathcal{P} est un plan solution, retourner P
2. analyse : appel des trois modules d'analyse :
DEFAUTS = propositions_non_expliquées(\mathcal{P})
\cup menaces(\mathcal{P})
\cup conflits_ressources(\mathcal{P})
3. sélection³ d'un défaut : $\Phi \in DEFAUTS$;
4. choix d'une résolvante : $\rho \in resolvantes(\Phi)$;
5. appel récursif : planifier(insertion(\mathcal{P}, ρ))

Figure 2. Algorithme général

Pour chaque défaut, un ensemble minimal de résolvantes est calculé. Pour un sous-but non expliqué, il s'agit de l'ensemble des événements du plan susceptibles d'expliquer le sous-but et de l'ensemble des tâches susceptibles d'expliquer le sous-but après leur insertion dans le plan [GHA 94]. Pour une menace, il s'agit d'un ensemble de contraintes temporelles ou portant sur les variables permettant de rendre la menace impossible. Pour un conflit de ressource, il s'agit d'un ensemble de contraintes temporelles d'ordonnancement permettant de limiter la consommation instantanée de ressource ; d'un ensemble de contraintes sur les variables représentant les ressources allouées ; et de l'ensemble des tâches susceptibles de produire la ressource après leur insertion dans le plan (voir le tableau de la figure 3).

	proposition non expliquée	menace	conflit de ressource
<i>contrainte temporelle</i>	précédence causale	promotion/démotion	ordonnancement
<i>contrainte sur variable</i>	co-désignation	séparation/co-désignation	restriction sur ressource allouée
<i>proposition temporelle</i>	lien causal		
<i>nouvelle tâche</i>	établissement		production

Figure 3. Typologie des défauts et des résolvantes

4.2 Contrôle et gestion des choix non-déterministes

L'algorithme général présente deux points de décision essentiels quant au parcours correct de l'espace de recherche et aux performances du système : la sélection d'un défaut et le choix d'une de ses résolvantes. Pour un défaut donné, le choix d'une de ses résolvantes s'effectue selon une stratégie de **moindre engagement**. La sélection

3. Suivant la terminologie définie dans [WEL 95] nous employons le mot *choix* pour désigner un choix non-déterministe et par conséquent un point possible de retour-arrière par opposition à *sélection* qui correspond à un choix heuristique sur lequel un retour-arrière ne sera pas nécessaire.

d'un défaut est basée sur la notion d'**opportunité** de la résolution dans un contexte donné.

4.2.1 La stratégie de moindre engagement

A un nœud donné de l'arbre de recherche, un plan partiel \mathcal{P} représente un ensemble (en général infini) de complétions possibles $COMP(\mathcal{P})$: tous les plans totalement instanciés atteignables à partir de \mathcal{P} par insertion de contrainte ou de tâche. L'insertion d'une contrainte ρ (sous la forme d'une résolvante) sur le plan partiel va éliminer un certain nombre de complétions possibles de sorte que $COMP(insertion(\mathcal{P}, \rho)) \subset COMP(\mathcal{P})$. Si l'on veut conserver un plan partiel le moins contraignant possible afin de pouvoir, ultérieurement dans la recherche, faire face à un éventail plus large de situations sans avoir à revenir en arrière, on a intérêt à choisir d'insérer en priorité les résolvantes qui éliminent le minimum de complétions de ce plan partiel. L'engagement lié à l'insertion d'une résolvante ρ sur un plan partiel \mathcal{P} est donc calculé comme une estimation du taux de complétions de \mathcal{P} éliminé par l'insertion de ρ :

$$engagt(\mathcal{P}, \rho) = 1 - \frac{card(COMP(insertion(\mathcal{P}, \rho)))}{card(COMP(\mathcal{P}))}$$

Pour une contrainte ρ donnée, l'engagement lié à son insertion dans le plan partiel est estimé de manière locale en fonction du domaine courant des variables sur lesquelles porte la contrainte.

Étant donné un défaut, le système choisit toujours d'insérer en priorité la résolvante de ce défaut qui minimise l'engagement.

4.2.2 Une recherche opportuniste

Il reste alors une sélection à faire quant au défaut à résoudre. Nous effectuons cette sélection de manière opportuniste en essayant de minimiser les risques de retour-arrière. Une des originalités essentielles du système est que, à un nœud donné de l'arbre de recherche, les défauts sont sélectionnés indépendamment de leur type (résolution de sous-but, de menace ou de conflit de ressource). Il n'y a pas, comme dans SNLP [MCA 91], d'ordre posé *a priori* en fonction du type des défauts (SNLP gère toujours les menaces en priorité avant de gérer l'explication des sous-buts) : dans IXTET, les défauts sont gérés, quelle que soit leur origine, en fonction de l'opportunité de leur résolution. Pour un défaut donné Φ , un critère $opp(\Phi)$ permet d'estimer la facilité qu'il y aura à faire un choix parmi les différentes résolvantes de Φ dans le cadre de la stratégie de moindre engagement définie dans le paragraphe précédent ; nous sélectionnons en priorité les défauts qui maximisent cette facilité. Si le défaut Φ admet les résolvantes $\{\rho_1, \dots, \rho_k\}$ et si ρ_{min} est la meilleure résolvante, $opp(\Phi)$ estime dans quelle mesure l'engagement lié à l'insertion de ρ_{min} est largement intéressant par rapport à celui lié aux autres résolvantes de Φ :

$$\frac{1}{opp(\Phi)} = \sum_{i=1}^k \frac{1}{1 + engagt(\mathcal{P}, \rho_i) - engagt(\mathcal{P}, \rho_{min})}$$

Il est à noter que l'on a toujours $0 < opp(\Phi) \leq 1$. $opp(\Phi) = 1$ si et seulement si Φ possède une seule résolvante, on dit dans ce cas que Φ est un défaut **déterministe** dans la mesure où il n'y aura aucun choix à effectuer. Les défauts déterministes sont bien entendu résolus en priorité. De manière générale, le critère opp permet de sélectionner en priorité les défauts qui ont peu de résolvantes, ceci étant modulé par la valeur de l'engagement lié à chacune des résolvantes.

4.2.3 Une recherche complète

L'arbre de recherche global est exploré grâce à un algorithme A_ϵ [GHA 83]. Notre objectif pour la recherche d'une solution est double : trouver une solution en un minimum de temps de calcul et trouver une solution de “bonne” qualité. Ces deux points ne sont toutefois pas incompatibles : dans le cadre d'une stratégie de moindre engagement (visant à améliorer les performances du système), les résolvantes choisies en priorité sont celles qui sur-constraignent le moins le plan partiel et, ce faisant, on aboutit à une solution peu contrainte et très souple dans le sens où elle représente une très grande variété d'instances possibles, et ceci correspond bien, en général, à la conception intuitive d'un “bon” plan.

L'estimation d'un nœud (f) correspond à la somme entre l'engagement lié à l'insertion de toutes les résolvantes depuis le nœud racine (g) et une estimation heuristique (h) de l'engagement minimal qu'il resterait à effectuer sur le plan partiel pour arriver à une solution. Bien que cette heuristique h , calculée en cumulant l'engagement lié aux meilleures résolvantes des défauts non-résolus du plan partiel, ne soit pas minoraute, on constate de manière empirique qu'elle est relativement efficace pour guider l'exploration de l'arbre de recherche.

Enfin, une caractéristique essentielle de notre système est sa **complétude** : s'il existe une solution au problème posé, alors le planificateur trouvera un plan solution et s'il n'en existe pas, il devra parcourir tout l'arbre de recherche avant de donner une réponse négative. A noter toutefois que, du fait de l'utilisation de variables (temporelles ou atemporelles) à domaine fini, l'arbre de recherche est lui-même de taille finie.

5 La hiérarchie de moindre engagement d' IXTET

L'application des techniques de hiérarchisation par abstraction présentées en début de cet article apparaît comme un moyen intéressant pour contrôler plus efficacement la recherche au sein du planificateur IXTET. Toutefois, du fait de la richesse de représentation d' IXTET , nous avons dû dans un premier temps étendre les résultats classiques de la hiérarchisation par abstraction. Nos travaux nous ont alors conduits à proposer un nouveau principe de génération automatique et de gestion de la hiérarchie qui étend au paradigme de hiérarchie d'abstraction l'approche de moindre engagement.

5.1 Hiérarchie d'abstraction pour IXTET

Nous venons de voir qu'une composante essentielle du contrôle de la recherche consiste à sélectionner le prochain défaut à résoudre dans le plan partiel courant.

Pour ce défaut, un ensemble minimal de résolvantes est calculé à partir duquel sont créés les noeuds successeurs dans l'arbre de recherche. Ainsi, la généralisation des travaux décrits plus haut sur le formalisme STRIPS nous conduit à associer un niveau d'abstraction à chacun des différents défauts du plan. Dans IXTET, nous avons vu que les défauts sont de trois types :

- sous-but pendant : par exemple,
`hold(INCUBE(Culture1,T20):true, (t_but,fin));`
- menace : entre `event(LOCALISATION(Rb1):(Banc,Mouvmnt), t)` et
`hold(LOCALISATION(Rb1):Banc, (t1,t2))` par exemple ;
- conflit de ressource, comme
`use(ROBOT(Rb1):1,(t1,t2))` et `use(ROBOT(Rb1):1,(t3,t4))`.

Pour chaque instance de défaut, un seul *nom d'attribut* est présent (ici : INCUBE, ROBOT ou LOCALISATION). Suite à cette observation et au fait que la complexité de l'algorithme de contrôle est alors peu augmentée, nous avons choisi, dans une première approche, de caractériser le niveau d'abstraction d'un défaut par le *nom de l'attribut* sur lequel il porte.

Hypothèse 1 (un unique niveau par nom d'attribut)

Un niveau d'abstraction représente un ensemble de défauts. Deux défauts portant sur le même nom d'attribut ont nécessairement même niveau d'abstraction.

Au niveau du contrôle, la gestion d'une hiérarchie d'abstraction consiste donc à résoudre les défauts dans les différents niveaux d'abstraction, partant des défauts les plus abstraits vers les défauts les moins abstraits. Nous avons choisi de définir des hiérarchies vérifiant une propriété similaire à l'OMP étendue à notre formalisme :

Propriété 2 (monotonie ordonnée pour IXTET)

Pour chaque plan partiel courant possible, aucun affinement envisageable de ce plan partiel suite à la résolution de défauts ne fera apparaître de nouveau défaut qui soit d'un niveau strictement supérieur au niveau d'abstraction courant.

Planifier dans le contexte d'une telle hiérarchie signifie que les seules causes de retour-arrière à partir d'un plan partiel sont l'absence de résolvante pour un défaut du niveau d'abstraction courant ou l'inconsistance des réseaux de contraintes sur les variables (temporelles ou atemporelles). L'efficacité d'une telle approche a été empiriquement prouvée dans de nombreux domaines.

Malgré l'extrême simplicité de ce critère, différents types de hiérarchies qui le vérifient sont possibles. Tous les systèmes de planification hiérarchiques (ABSTRACTIPS [SAC 74], ABTWEAK [YAN 90], PRODIGY[KNO 94]) imposent un ordre total pré-établi sur les niveaux d'abstraction ; plus précisément, la hiérarchie qu'ils utilisent est une partition calculée hors-ligne de l'ensemble des défauts possibles d'un plan

partiel, chaque sous-ensemble de cette partition correspondant à un niveau. Dans ces approches, une telle décomposition s'effectue dans un premier temps en recherchant un ordre partiel sur les niveaux satisfaisant l'OMP (cf. propriété 1) puis, dans un deuxième temps, par la linéarisation de cet ordre partiel en fonction d'une stratégie définie hors-ligne [KNO 94, BAC 94]. Cette approche soulève un important problème : comme cette stratégie est statique et décorrélée du processus de planification, elle conduit à un engagement *a priori* qui peut être préjudiciable aux performances de l'algorithme de planification. Il peut ainsi en résulter une augmentation du nombre de retours-arrière entre niveaux d'abstraction, ou une dégradation de la qualité des solutions obtenues. Ce dernier risque est clairement exprimé sur la base d'un exemple dans [BÄC 95].

C'est pourquoi nous proposons une solution originale : dans **I_XTET**, la hiérarchie ne sera totalement définie qu'au cours du processus de planification. Les différents niveaux d'abstraction sont construits en ligne en fonction de l'évolution de la recherche. Si L représente le niveau d'abstraction courant, c'est-à-dire l'ensemble des défauts qui sont couramment analysés sur ce plan partiel, l'idée principale est de ne pas attendre que tous les défauts de L aient été résolus avant de faire évoluer le niveau d'abstraction de L à L' comme cela est fait dans les approches classiques.

5.2 Une approche de moindre engagement

La question qui se pose maintenant est de savoir comment construire automatiquement une telle hiérarchie pour **I_XTET**. A la manière d'**ALPINE**, notre approche repose sur la définition de conditions suffisantes extraites de la description syntaxique des tâches du domaine. Ces conditions s'expriment par des contraintes sur les niveaux d'abstraction et les hiérarchies d'abstraction possibles.

Suite à notre choix de représentation des niveaux hiérarchiques en termes d'ensembles de défauts portant sur un même nom d'attribut, ces contraintes sur les niveaux d'abstractions sont représentées par une relation d'ordre partiel sur les différents noms d'attributs.

Dans la définition qui suit, nous supposons que toutes les sous-tâches d'une tâche donnée ont déjà été récursivement développées si bien qu'une tâche ne contient que des ensembles d'événements, d'assertions, d'utilisations de ressources et des contraintes d'instanciations sur les variables, temporelles ou non, qu'elle utilise. Nous notons att_p le nom de l'attribut sur lequel porte une proposition temporelle p (qu'il s'agisse d'un événement, d'une assertion ou d'une utilisation de ressource) et att_Φ le nom de l'attribut sur lequel porte un défaut quelconque Φ .

Définition 6 (contraintes \prec sur les noms d'attributs)

Pour chaque tâche \mathcal{T} du domaine, si e est un événement (event) ou une proposition de production de ressource (produce) et si p est une proposition temporelle quelconque de \mathcal{T} , alors $att_p \prec att_e$. Nous identifierons par la suite \prec et sa fermeture transitive.

Afin de mieux comprendre la nécessité de ces conditions, il est utile de revenir à la définition classique de ces contraintes pour une représentation de type STRIPS

(propriété 1). Dans ce cas, l'idée est de repousser le traitement des propositions de type précondition après le traitement des propositions de type effet. Pour cela, on place dans un même niveau les différents effets d'un même opérateur, et dans des niveaux inférieurs les différentes préconditions. Dans le formalisme IXTET les effets sont modélisés par des événements, changements de valeurs d'un attribut d'état, et par des productions de ressource. La notion de précondition peut alors être associée à toute proposition temporelle présente dans la tâche. Poser les contraintes $att_p \prec att_e$ permet alors comme dans la propriété 1 de mettre dans un même niveau les différents attributs d'effets (au sens général) de la tâche, et dans des niveaux moins abstraits les différents attributs préconditions (au sens général) de la tâche.

Ainsi, la présence de la contrainte $att \prec att'$ sera utilisée pour interdire au cours du processus de planification, de résoudre un défaut sur un nom d'attribut att dans un niveau plus abstrait (i.e. avant) qu'un défaut portant sur un attribut att' . En d'autres termes, pour pouvoir résoudre un défaut sur att , il sera nécessaire d'attendre que tous les défauts sur att' aient été résolus.

Il est alors utile à partir de l'ensemble des contraintes \prec de définir une relation $<$ entre ensembles de noms d'attributs qui sont \prec -équivalents (nous identifions ici $<$ et sa fermeture transitive) :

Définition 7 (ordre partiel $<$)

Soit $[att]$ la classe de \prec -équivalence du nom d'attribut att :
 $[att] = \{att' \mid att \prec att' \wedge att' \prec att\}$. On définit alors l'ordre partiel $<$ entre classes :
 $[att] < [att']$ si et seulement si $[att] \neq [att']$ et $att \prec att'$.

Cet ordre partiel $<$ peut être représenté par un graphe de contraintes orienté et acyclique \mathcal{G} dont les nœuds sont des ensembles de noms d'attributs \prec -équivalents et les arcs représentent les contraintes de $<$ sur les niveaux d'abstractions.

Pour illustrer ces définitions, considérons la tâche ALLER_A décrite précédemment. Des contraintes sur les niveaux d'abstraction induites par cette tâche sont :

ROBOT \prec LOCALISATION; PUISSANCE \prec LOCALISATION; COMMUNICATION \prec LOCALISATION; ZONE_DEPLACEMENT \prec LOCALISATION.

La figure 4 présente un graphe \mathcal{G} complet pour le domaine des expériences de biologie embarquées donné en annexe. Sur le graphe, un arc issu d'une classe $[att]$ vers la classe $[att']$ signifie que tous les défauts sur des noms d'attributs de $[att]$ doivent nécessairement être résolus avant ceux portant sur un nom d'attribut de $[att']$, c'est-à-dire : $[att'] < [att]$. On constate que dans cet exemple, les classes de noms d'attributs \prec -équivalents sont réduits à des singuliers.

Une fois les contraintes sur l'ordre des noms d'attributs exprimées, nous devons expliciter une hiérarchie d'abstraction vérifiant la propriété de monotonie ordonnée d'IXTET .

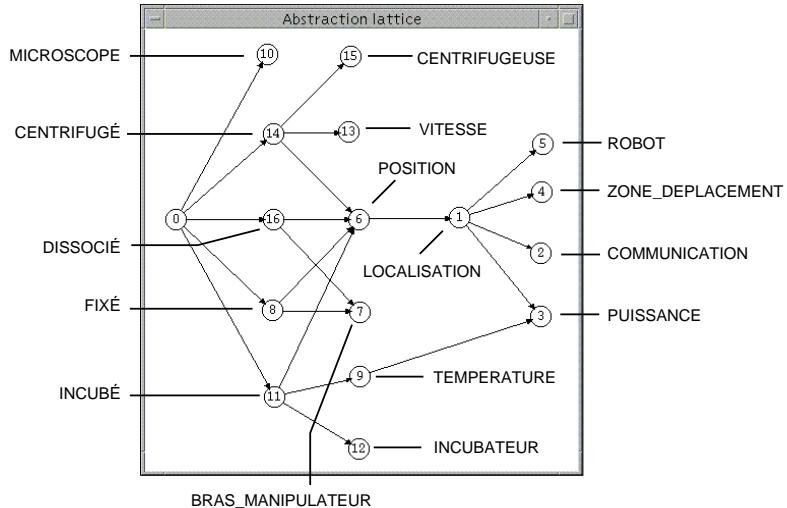


Figure 4. Graphe \mathcal{G} pour le domaine des expériences de biologie

Dans IXTE_T, nous gérons directement la hiérarchie partiellement ordonnée au sein du processus de planification. C'est dans cette perspective que nous définissons la notion d'*état d'abstraction*:

Définition 8 (état d'abstraction)

Un état d'abstraction est un couple (R, C) ⁴ d'ensembles de noms d'attributs, où $R = \{att_{s1}, \dots, att_{sn}\}$ représente l'ensemble des attributs ayant été complètement résolus jusqu'ici, et $C = \{att_{c1}, \dots, att_{cm}\}$, l'ensemble des attributs qui sont en cours de résolution.

Un état d'abstraction décrit, pour un nœud donné de l'arbre de recherche, le degré d'abstraction du plan courant. A un nœud donné, seuls les défauts sur des noms d'attributs de C sont examinés. L'ensemble C de l'état d'abstraction courant caractérise donc le niveau d'abstraction courant L :

$$L = \{\Phi \mid att_\Phi \in C\}$$

Moyennant ces définitions, définir une hiérarchie d'abstraction se ramène donc à définir la manière dont évoluent les états d'abstraction le long des nœuds de l'arbre de recherche.

Dans le nœud racine correspondant au scénario initial, l'état d'abstraction est défini comme (R_0, C_0) , où $R_0 = \emptyset$, et $C_0 = \{att \mid \{att' \mid [att] < [att']\} = \emptyset\}$ c'est-à-dire les noms d'attributs les plus abstraits au sens de l'ordre partiel $<$ (éléments maximaux). Supposons alors que durant l'étape de résolution des défauts de

4. Nous notons R pour attributs Résolus et C pour attributs en Cours de résolution.

L_0 , un ensemble $[att]$, avec $att \in C_0$, de noms d'attributs \prec -équivalents ait été complètement résolu. Cela signifie que le plan partiel ne contient plus aucun défaut sur des noms d'attributs de $[att]$. Nous sommes sûrs, d'autre part, que de tels défauts n'apparaîtront pas dans la suite du processus ; en effet, la définition de C_0 interdit l'existence de défauts Φ de L_0 dont la résolution conduirait à l'apparition de nouveaux défauts Φ' , avec $att_{\Phi'} \in [att]$.

Puisque aucun défaut sur les noms d'attributs \prec -équivalents de $[att]$ ne pourra apparaître, ces attributs peuvent être ôtés de l'ensemble C de l'état d'abstraction courant. L'approche hiérarchique classique consisterait à attendre que l'ensemble C soit complètement vide avant de calculer un nouvel ensemble C' et un nouveau niveau d'abstraction L' . En fait, cela n'est pas nécessaire et nous pouvons immédiatement ajouter à C certains nouveaux noms d'attributs à traiter à partir du moment où ceci ne viole pas les contraintes \prec qui assurent l'OMP.

La règle générale de mise à jour du niveau d'abstraction courant (R, C) à un nœud de l'arbre de recherche est la suivante :

Définition 9 (mise à jour des niveaux d'abstraction)

Lorsqu'un ensemble $[att_{résolu}]$ de noms d'attributs \prec -équivalents s'avère être complètement résolu, $[att_{résolu}]$ est ôté de C , rajouté à R et C est recalculé à partir du nouvel ensemble d'attributs résolus R par :

$$\begin{aligned} R' &= R \cup [att_{résolu}] \\ C' &= C - [att_{résolu}] \\ &\quad \cup \{att_{nouveau} \mid [att_{nouveau}] < [att_{résolu}] \wedge \{att \mid [att_{nouveau}] < [att]\} \subset R'\} \end{aligned}$$

Un exemple de mise à jour d'un état d'abstraction au cours du processus de planification est montré sur la figure 5 correspondant au nœud de l'arbre de recherche où le dernier défaut sur l'attribut INCUBE a été résolu.

Comme pour l'approche classique, on montre le théorème suivant :

Théorème 1

Les hiérarchies d'abstraction utilisées dans IXTEIT vérifient la propriété de monotonie ordonnée (propriété 2).

Ce théorème (voir démonstration en annexe) indique donc que pour chaque plan partiel courant possible, aucun affinement envisageable suite à la résolution de défauts ne fera apparaître de nouveau défaut qui soit d'un niveau strictement supérieur au niveau d'abstraction courant.

La différence principale avec l'approche classique tient dans le fait que les niveaux que nous générions ne forment pas une partition de l'ensemble des défauts : généralement, deux niveaux d'abstraction consécutifs s'intersectent. L'avantage de notre approche réside dans l'accroissement de l'opportunisme du contrôle qui ne prend en compte que les contraintes d'abstraction entre attributs qui sont directement issues

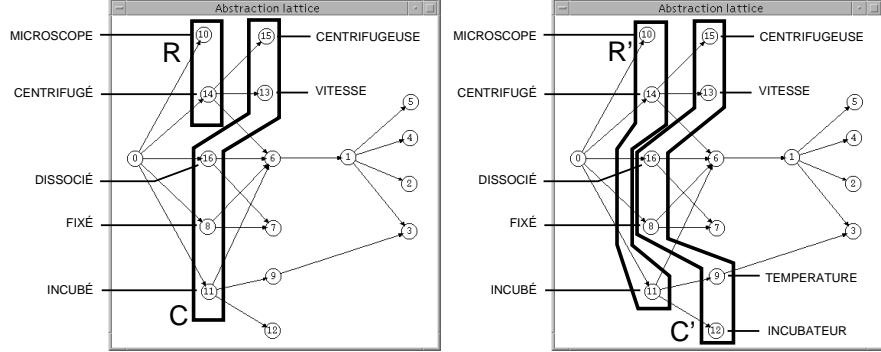


Figure 5. Deux états d'abstraction consécutifs pour le domaine des expériences de biologie

opérateurs	préconditions	effets
s_{2i}	$\{p_{2i-1}, p_{2i-2}\}$	$\{p_{2i}\}$
r_{2i}	$\{p_{2i-1}, p_{2i-2}\}$	$\{\neg p_{2i}\}$
s_{2i+1}	$\{\neg p_{2i-1}, \neg p_{2i-2}\}$	$\{p_{2i+1}\}$
r_{2i+1}	$\{\neg p_{2i-1}, \neg p_{2i-2}\}$	$\{\neg p_{2i+1}\}$

Table 1. Description des opérateurs pour $0 \leq 2i < n$. s_0, r_0, s_1 et r_1 ont des préconditions vides

du graphe \mathcal{G} . Il est ainsi possible de développer en priorité les branches parallèles du graphe \mathcal{G} les plus opportunes pour la recherche du plan et d'attendre d'être dans un contexte plus complet pour avancer dans les branches moins opportunes.

Afin d'illustrer la spécificité de notre approche, considérons l'exemple introduit dans [BÄC 95]. Le domaine considéré est décrit par les n (pair) propositions p_0, \dots, p_{n-1} , et par les $2n$ opérateurs $s_0, r_0, \dots, s_{n-1}, r_{n-1}$ décrits tableau 1.

Partant de l'état initial $\neg p_0, \dots, \neg p_{n-1}$, le problème de planification consiste à établir p_{n-2} et p_{n-1} . Les contraintes d'abstraction entre propositions sont ici $p_{2i-3} \prec p_{2i-1}, p_{2i-4} \prec p_{2i-1}, p_{2i-3} \prec p_{2i-2}, p_{2i-4} \prec p_{2i-2}$, pour $i = 2, \dots, n$. Les deux hiérarchies considérées dans [BÄC 95] sont deux des ordres totaux compatibles avec ces contraintes :

$$\begin{aligned} \mathcal{H}_1 &= (\{p_0\}, \{p_1\}, \{p_2\}, \dots, \{p_{n-2}\}, \{p_{n-1}\}) \\ \mathcal{H}_2 &= (\{p_1\}, \{p_0\}, \{p_3\}, \dots, \{p_{n-1}\}, \{p_{n-2}\}) \end{aligned}$$

Pour un algorithme de planification hiérarchique de type linéaire (c'est-à-dire tel que tous les opérateurs d'un plan solution au niveau d'abstraction N sont nécessairement totalement ordonnés en passant au niveau suivant $N - 1$), il est alors montré dans [BÄC 95] que \mathcal{H}_1 conduit à une solution de taille n , alors que \mathcal{H}_2 mène nécessairement à une solution de taille $2^{\frac{n}{2}+1} - 2$. Ainsi le choix de la hiérarchie est crucial

quant à la qualité de la solution obtenue, et à la longueur du chemin permettant de l'atteindre.

Bien que cette argumentation repose fondamentalement sur l'emploi d'un planificateur hiérarchique **linéaire**, et qu'un algorithme comme le nôtre - ou plus simplement de type ABTWEAK [YAN 90] gérant un ordre partiel entre opérateurs - n'aurait pas de problème pour trouver le plan le plus court, que ce soit avec \mathcal{H}_1 ou \mathcal{H}_2 , l'intérêt de notre hiérarchisation dynamique peut encore être illustré ici. En effet, indépendamment du type de planificateur (linéaire ou non), notre approche nous permet d'entrelacer dynamiquement les choix entre \mathcal{H}_1 et \mathcal{H}_2 , en conservant la structure $\{p_0, p_1\}, \{p_2, p_3\}, \dots, \{p_{n-2}, p_{n-1}\}$. A chaque étape, il faut choisir entre p_{2i} et p_{2i+1} , mais il faut traiter les deux avant de passer au couple $\{p_{2i-2}, p_{2i-1}\}$. Dès lors, le fait que notre approche nous permette de trouver le plan le plus court repose entièrement sur le type de contrôle utilisé par le planificateur. Si, ayant le choix entre p_{2i} et p_{2i+1} , le planificateur choisit systématiquement de traiter p_{2i} avant p_{2i+1} , alors on retombe sur la hiérarchie \mathcal{H}_2 et le plan exponentiellement long. Si par contre le contrôle utilise une heuristique intelligente de type "planifier p_{2i+1} avant p_{2i} ", alors la solution de taille n pourra être générée. Ainsi, dans ce cadre là, notre approche hiérarchique n'est pas affectée par l'argument de [BÄC 95] car elle ne surconstraine pas l'ensemble des propositions, et laisse au contrôle le soin de choisir entre les différents sous-but non contraints entre eux.

Il est possible de borner le nombre des différents niveaux d'abstraction que notre hiérarchisation dynamique peut engendrer. Nous définissons la profondeur d'abstraction $d_{\mathcal{G}}$ comme le nombre de sommets du plus long chemin sur le graphe \mathcal{G} . Soit d'autre part $n_{\mathcal{G}}$ le nombre de noeuds de \mathcal{G} . Le résultat suivant, montré en annexe, caractérise les hiérarchies d'abstraction indépendamment du problème spécifique traité :

Théorème 2

Le nombre maximal l des différents niveaux d'abstraction parcourus au cours de la recherche depuis le scénario initial \mathcal{P}_0 jusqu'à un plan solution \mathcal{P} est tel que $d_{\mathcal{G}} \leq l \leq n_{\mathcal{G}}$.

6 Quelques améliorations de la hiérarchie

6.1 Effets primaires et secondaires

Pour les mêmes raisons que celles décrites par Knoblock [KNO 94] et davantage formalisées dans [FIN 95], il peut être intéressant de faire la distinction entre *effets primaires* et *effets secondaires* d'une tâche. Dans le contrôle du système de planification IXTE, une tâche peut être insérée au plan partiel dans le but de résoudre une proposition d'état non expliquée ou de produire une ressource susceptible d'intervenir dans un conflit (voir tableau 3). La tâche va permettre de résoudre le défaut si elle contient un effet, qu'il s'agisse d'un événement (*event*) ou une production de ressource (*produce*), adéquat. Souvent, néanmoins, une tâche donnée ne pourra être insérée dans le plan que pour un sous-ensemble de ses effets.

Considérons ainsi la tâche `INSTALLER` décrite en annexe. Elle consiste pour un robot $?r$ à installer un élément $?El_t$ dans une zone $?Zone$. Ses effets sont de modifier la position de l'objet $?El_t$ ainsi que la localisation du robot qui se trouvera dans la zone $?Zone$ après exécution de la tâche. Toutefois, il semble inutile d'envisager l'insertion dans le plan d'une telle tâche pour établir le fait que le robot est localisé dans une zone particulière. Nous considérerons donc l'attribut `LOCALISATION` comme un attribut secondaire de la tâche, tandis que l'attribut `POSITION` sera, lui, primaire.

En distinguant de cette manière attributs primaires et attributs secondaires, il sera possible d'éliminer de \mathcal{G} certaines contraintes qui n'ont pas raison d'être étant donnée notre manière de gérer les effets des tâches dans le processus de planification.

L'avantage de cette distinction est alors double : en plus d'une réduction du facteur de branchement dans l'arbre de recherche (il y aura moins de résolvantes pour les défauts de type "proposition non-expliquée" et "conflit de ressource"), cela permettra d'obtenir une hiérarchie moins contraite ce qui donnera plus de liberté au contrôle de la recherche pour appliquer sa stratégie de recherche opportuniste. La nouvelle définition de l'ordre hiérarchique partiel \prec devient alors :

Définition 10 (contraintes \prec_{PS} sur les noms d'attributs) ⁵

*Pour chaque tâche \mathcal{T} du domaine, si e est un événement (*event*) ou une proposition de production de ressources (*produce*) intervenant comme effet primaire de \mathcal{T} et si p est une proposition temporelle quelconque de \mathcal{T} , alors $att_p \prec_{PS} att_e$. Nous identifierons par la suite et sauf indication contraire \prec_{PS} et sa fermeture transitive.*

6.2 Prise en compte du problème à résoudre

Dans [KNO 94], l'auteur montre que de meilleures hiérarchies d'abstraction peuvent être obtenues en prenant en compte le problème de planification que l'on cherche à résoudre lors du calcul du graphe de contraintes \mathcal{G} . Le principe est le même que dans le cas de la distinction effet primaire/effet secondaire que nous venons de décrire : il s'agit d'éviter d'insérer sur le graphe \mathcal{G} des contraintes inutiles et d'obtenir ainsi une caractérisation plus générale des hiérarchies admissibles au sens de l'OMP. Nous montrons dans ce paragraphe comment on peut généraliser cette approche au formalisme IXTET. Nous noterons toutefois que si les prédicts *produce* ne sont pas gérés⁶, la prise en compte du problème n'entraîne aucune amélioration de la hiérarchie.

Soit e une proposition temporelle de type *event* ou *produce* appartenant à une tâche \mathcal{T} . Nous dirons que e est **pertinente** pour le scénario initial \mathcal{P}_0 , et nous noterons $e \in \text{pertinent}(\mathcal{T}, \mathcal{P}_0)$, si la proposition e peut justifier l'insertion de la tâche \mathcal{T} dans le plan partiel pour résoudre un défaut du scénario initial ou un nouveau défaut que pourrait engendrer l'insertion dans le plan d'une tâche \mathcal{T}' , avec $\text{pertinent}(\mathcal{T}', \mathcal{P}_0) \neq \emptyset$.

Nous dirons alors qu'un attribut att est **pertinent pour le scénario initial** s'il existe une tâche \mathcal{T} et une proposition e dans \mathcal{T} de type *event*, ou *produce*, portant sur

5. Nous notons PS pour Primaire/Secondaire.

6. En d'autre termes, si l'on ne gère que *event*, *hold* et *use*.

l’attribut att et telle que $e \in pertinent(\mathcal{T}, \mathcal{P}_0)$.

Il est clair, d’après cette définition récursive des attributs pertinents pour le scénario initial, qu’au cours de la résolution les seules tâches qui pourront être insérées dans le plan sont telles que $pertinent(\mathcal{T}, \mathcal{P}_0) \neq \emptyset$.

La façon naturelle de définir les conditions suffisantes pour que la hiérarchie vérifie l’OMP devient alors :

Définition 11 (contraintes \prec_{SI} sur les noms d’attributs) ⁷

Pour chaque tâche \mathcal{T} du domaine, si e est un événement (event), une proposition de production de ressources (produce) tel que $e \in pertinent(\mathcal{T}, \mathcal{P}_0)$ et si p est une proposition temporelle quelconque de \mathcal{T} , alors $att_p \prec_{SI} att_e$. Nous identifierons par la suite \prec_{SI} et sa fermeture transitive.

Dans le cas particulier où aucun prédicat de type *produce* n’intervient dans la description du domaine, nous montrons le théorème suivant (voir annexe) qui stipule que les ordres partiels \prec et \prec_{SI} sont identiques et donc que la prise en compte du scénario initial n’apporte aucun gain sur la hiérarchie générée.

Théorème 3

*En l’absence de prédicats de type *produce*, $\forall \mathcal{P} \in COMPL(\mathcal{P}_0)$, $\forall \Phi, \Phi'$ défauts de \mathcal{P} ,*

$$att_\Phi \prec att_{\Phi'} \Rightarrow att_\Phi \prec_{SI} att_{\Phi'}$$

Rappelons que $COMPL(\mathcal{P}_0)$ désigne l’ensemble des plans partiels \mathcal{P} susceptibles d’être développés au cours de la recherche.

Il est intéressant de savoir que la preuve de cette propriété repose sur le fait que les seuls effets possibles d’une tâche consistent en des événements ; lesquels événements expriment à la fois un effet et une condition sur la tâche (le fait que l’ancienne valeur nécessite d’être expliquée).

Nous avons considéré dans [LAB 95a] un nouveau type de prédicat, *assert*, proche de *produce*, qui exprime typiquement des effets qui ne sont associés à aucune condition particulière. En pratique, $assert(att(x_1, \dots, x_n) : v, t)$ permet de modéliser le changement de valeur de l’attribut d’état $att(x_1, \dots, x_n)$ d’une ancienne valeur inconnue à la nouvelle valeur v . On peut alors montrer comme dans l’exemple suivant que dans le cas où des prédicats de type *produce* ou *assert* sont présents la prise en compte du scénario initial peut permettre de relâcher la hiérarchie.

Supposons le cas d’un robot dans une pièce capable d’effectuer des tâches comme le perçement d’une saignée dans le mur (`PERCE_SAIGNEE`), l’installation d’une tuyauterie dans la saignée afin d’installer l’eau courante (`INSTALLE_EAU`) et de nettoyer les murs (`LAVE_MUR`).

La formalisation de ces trois tâches est donnée en figure 6 ainsi qu’un petit scénario initial où, au départ, le mur est propre, l’eau non-installée et dont l’objectif est l’installation de l’eau.

7. Nous notons SI pour Scénario Initial.

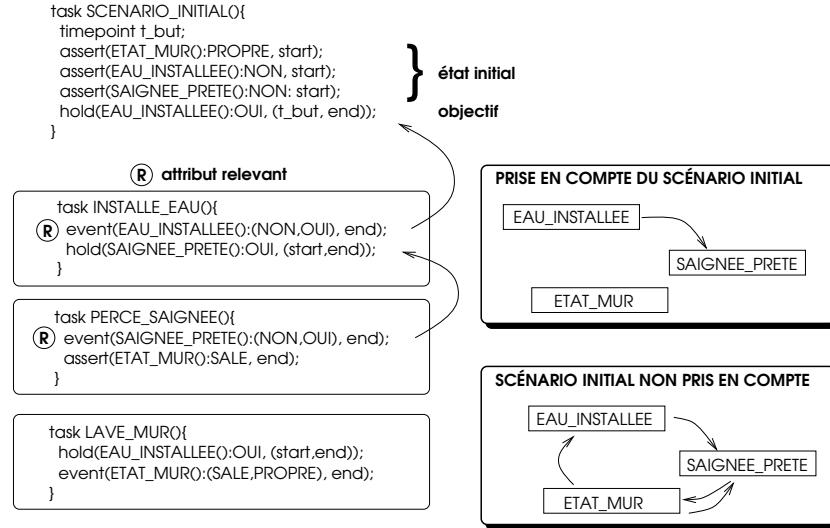


Figure 6. Une hiérarchie dépendant du scénario initial

7 Implémentation, résultats et limitations

7.1 Génération automatique de la hiérarchie

L’approche que nous venons de décrire, à l’exception de la prise en compte du scénario initial, § 6.2, a été implémentée sur le système de planification IXTET . Un *module de génération automatique des niveaux d’abstraction* permet de construire le graphe \mathcal{G} en analysant la description syntaxique des tâches et en ne posant que l’ensemble des contraintes suffisantes à assurer l’OMP. L’utilisateur a ensuite, s’il le souhaite, la possibilité de sur-constrainer cette structure hiérarchique.

Les contraintes \prec_{PS} sont représentées comme un treillis IXTET [GHA 89] où chaque noeud représente un ensemble de noms d’attributs devant nécessairement être géré dans le même niveau d’abstraction. Ce treillis est initialisé avec autant de noeuds non contraints qu’il y a de noms d’attributs différents sur le domaine. Les tâches qui sont pertinentes pour le domaine sont alors parcourues récursivement et les conditions suffisantes pour satisfaire l’OMP sont posées sur le treillis. A noter que nous avons implémenté la notion d’effets primaires/secondaires d’une tâche : une tâche ne pouvant être insérée dans le plan que pour utiliser ses effets primaires, seuls ses effets primaires doivent être pris en compte lorsque l’on génère la hiérarchie d’abstraction (cf. définition 10). Lorsque l’insertion d’une contrainte fait apparaître un cycle sur le graphe orienté, les noeuds de ce cycle sont fusionnés en un unique noeud. L’efficacité de la propagation des contraintes et des requêtes sur un treillis IXTET (ces deux procédures ont une complexité linéaire en moyenne) permettent une complexité globale du processus de génération de la hiérarchie en $O(m.n.s.e)$ si m est le nombre de modèles de tâches, n le nombre de noms d’attributs du domaine, s la taille moyenne des tâches

en termes de propositions temporelles et e le nombre moyen d'effets primaires d'une tâche (en général, $e = 1$ ou 2). Un exemple de hiérarchie automatiquement générée est celui de la figure 4.

7.2 Gestion de la hiérarchie d'abstraction

Comme nous l'avons décrit plus haut, nous associons à chaque nœud de l'arbre de recherche un état d'abstraction local (R, C). Lorsqu'un nom attribut att de C se retrouve vierge de tout défaut, une vérification est effectuée pour voir si l'ensemble $[att]$ des noms d'attributs \prec_{PS} -équivalents à att sont aussi résolus ; Dans ce cas, le niveau d'abstraction courant est modifié selon la définition 9 (voir illustration sur la figure 5).

A un nœud donné de la recherche, si n est le nombre de symboles d'attributs et p_G le degré de parallélisme maximal⁸ de \mathcal{G} , la complexité supplémentaire du fait de la prise en compte de la hiérarchie d'abstraction est de l'ordre de $O(n.p_G)$ dans les cas où le niveau d'abstraction est modifié, $O(n)$ sinon. Cette complexité reste toujours négligeable devant la complexité de l'analyse du plan partiel. A noter que pour calculer un nouvel état d'abstraction, nous n'utilisons pas, pour des raisons de complexité, l'algorithme qui pourrait naturellement être suggéré par la définition 9 (et qui demanderait d'exploiter toute la fermeture transitive de \prec) mais une définition strictement équivalente reposant uniquement sur l'ensemble des contraintes \prec effectivement posées lors de l'analyse des tâches.

Notre approche a été validée sur de nombreux domaines comme :

- la planification d'expériences de biologie à bord d'une station spatiale de type Columbus. Ce domaine a été modélisé sous IXET dans le cadre du projet PADRE du MESR, en collaboration avec les sociétés IXI et Matra Marconi Space France (MMS). Une description de cette application est donnée en annexe.
- le contrôle au niveau tâche d'une équipe de robots mobiles (Hilares, Iares). L'application Hilare se situe en environnement intérieur et vise à planifier et à coordonner les activités de déplacement et d'acquisitions d'informations de trois robots. L'application Iares, développée en collaboration avec le CNES se situe dans le domaine de la robotique autonome d'exploration planétaire sur la planète Mars. Il s'agit de planifier les déplacements et un certain nombre d'actions (acquisition d'échantillons du sol, cartographie de la zone, envoi d'informations à l'orbiteur, etc).
- l'ordonnancement de l'intégration de cases à équipements pour le lanceur Ariane 4. Ce problème nous a été soumis par Matra Marconi Space. Le lanceur Ariane 4 contient 8 cases à équipements destinées à recevoir les satellites. L'intégration d'une case donnée nécessite l'exécution d'une séquence de tâches ; ces tâches étant contraintes par des dates au plus tôt (arrivée des structures, de l'équipement) et des dates au plus tard (signatures de contrats, départ des bateaux pour

⁸. Ce degré étant défini comme la taille d'un ensemble indépendant maximal pour l'ordre partiel $<$.

Domaines	facteur de gain	$d_{\mathcal{G}}$	$n_{\mathcal{G}}/p_{\mathcal{G}}$	nombre attributs	nombre tâches	taille plan
<i>Hilares</i>	1.4	2	1.5	3	3	10
<i>Lares</i>	2.7	3	1.5	5	11	19
<i>Columbus</i>	prob. 1	1.7	4	1.7	16	11
	prob. 2	2.3	4	1.7	16	11
<i>Ariane</i>	3.9	6	6	6	0	56
<i>Finition pièce</i>	8.8	6	4	18	12	18

Table 2. Gains globaux liés à la hiérarchisation

	hiérarchie ordonnée hors-ligne :	hiérarchie dynamique :
<i>Nœuds</i>	305	59
<i>Retour-arrières</i>	14	0
<i>Temps CPU</i>	14.1 s	2.8 s

Table 3. Gains liés à l'approche de moindre engagement

Kourou). Chacune de ces tâches s'étale sur plusieurs jours de travail et nécessite, en concurrence avec les autres, diverses ressources humaines et moyens techniques. La structure très contrainte de ce problème privilégie une approche opportuniste telle celle utilisée par IXTEt par rapport à des approches plus classiques issues de la recherche opérationnelle. Comme l'on ne dispose pas de tâches à insérer puisque le plan donné en entrée les contient déjà toutes, la hiérarchie ne peut être générée automatiquement. Dans notre test, elle est donnée en entrée par l'utilisateur.

- la planification des travaux de finition d'une pièce (installation eau, électricité, papier peint) inspirée de [MIS 91].

Les principaux résultats sont résumés sur les tableaux 2 et 3. Le tableau 2 donne pour chaque domaine : le facteur de gain (défini comme le rapport entre le temps CPU sans utiliser d'abstraction et celui en utilisant notre approche de hiérarchie dynamique), la profondeur du graphe d'abstraction, le rapport $n_{\mathcal{G}}/p_{\mathcal{G}}$ (où $p_{\mathcal{G}}$ désigne le degré de parallélisme maximal de \mathcal{G}), le nombre d'attributs du domaine, le nombre de tâches modélisées et la taille du plan solution (donnée en nombre de tâches).

Deux conséquences importantes peuvent se dégager de l'étude que nous avons menée sur la hiérarchisation. Ces résultats ont été montrés dans le cadre du système IXTEt mais, dans la mesure où ils reposent essentiellement sur l'approche de moindre engagement, ils restent aussi valables pour la majorité des systèmes de planification utilisant une telle approche.

1. Comme seulement un sous-ensemble des défauts est examiné, l'analyse du plan partiel à chaque nœud de l'arbre de recherche est plus rapide que sans l'utili-

sation de hiérarchie. Le temps moyen passé à un nœud est globalement proportionnel au nombre de noms d'attributs présents dans C et donc, il dépend de la topologie du graphe \mathcal{G} . La profondeur d'abstraction $d_{\mathcal{G}}$ et le rapport $n_{\mathcal{G}}/p_{\mathcal{G}}$ entre la taille du graphe d'abstraction et son degré maximal de parallélisme sont deux facteurs qui peuvent permettre d'estimer le gain possible. Dans nos exemples, le facteur de gain varie de 1.4 (Hilares) à 8.8 (Finition pièce).

2. La hiérarchisation permet une structuration de l'espace de recherche qui conduit, en général, à une réduction des risques de retour-arrière dans la mesure où les caractéristiques les plus structurantes du plan sont examinées et résolues en premier. Notre approche originale de moindre engagement permet une exploitation encore plus poussée de cette propriété. Le tableau 3 illustre ceci à partir d'un problème très contraint du domaine IARES où notre approche dynamique trouve une solution sans retour-arrière et en environ 5 fois moins de temps qu'en utilisant une hiérarchie classique.

D'un autre côté, la hiérarchisation de la recherche signifie que le contrôle va se focaliser sur un niveau d'abstraction donné et "faire abstraction" des niveaux plus concrets. Une conséquence est que l'heuristique h mesurant la distance entre le nœud courant et un nœud solution sera moins informée et moins fiable pour indiquer au module de contrôle sur quel nœud revenir en arrière le cas échéant (nous pourrions bien entendu prendre en compte tous les défauts du plan partiel uniquement dans le calcul de h , mais cela reviendrait à perdre le gain important que nous avons déjà mentionné lié au nombre de défauts examinés). Pour résumer, la hiérarchisation réduit les risques de retour-arrière, mais lorsqu'un retour arrière survient, elle complexifie le problème du choix du nœud où revenir.

8 Conclusion

Nous avons vu comment un critère de hiérarchie (la propriété de monotonie ordonnée) initialement défini sur un formalisme de type STRIPS pouvait être étendu à une représentation beaucoup plus riche prenant en compte le temps et les ressources et où le besoin d'une structuration de la recherche est, de ce fait, encore plus fort.

En respectant la stratégie de moindre engagement utilisée dans IXTE^T (et dans la majorité des algorithmes de planification récents), nous avons développé une hiérarchie dynamique dépendant non seulement du problème à résoudre mais aussi de l'état courant du plan partiel en cours de planification. Seules les contraintes nécessaires à la vérification de l'OMP sont prises en compte ; aucune contrainte *a priori* n'est insérée afin d'aboutir à un ordre total.

L'intégration de notre approche hiérarchique au système de planification est schématisée sur la figure 1. L'intérêt de cette hiérarchisation de la recherche, en termes de réduction des risques de retour-arrière et d'efficacité globale a été clairement établi dans différents domaines.

Une question importante qui reste posée, concerne la démonstration théorique de l'intérêt d'une propriété telle que l'OMP et du contrôle associé. Certains résultats

intéressants ont déjà été obtenus par exemple dans [BAC 94], ou dans le domaine des CSP pour les problèmes de sélection de variables, sélection qui est le pendant CSP de la sélection d'un défaut [PUR 83], mais aucune analyse complète n'a été proposée à ce jour.

Les recherches futures dans ce domaine devraient consister à raffiner la granularité de notre hiérarchie pour la gérer non plus au niveau des noms d'attributs mais au niveau des attributs partiellement instanciés. Il est aussi envisageable d'exploiter la dimension temporelle du formalisme IXTET . Cela reviendrait à associer un niveau d'abstraction à un défaut en fonction des attributs partiellement instanciés qui le composent et de sa position temporelle dans le plan partiel. Ces développements devraient renforcer l'idée que les techniques courantes d'abstraction utilisées en planification sont plus liées à une stratégie de sélection du prochain défaut à résoudre qu'à une stratégie globale de résolution hiérarchique du problème.

9 Bibliographie

Références

- [BAC 91] BACCHUS F. et YANG Q., The downward refinement property. In : *Proceedings IJCAI-91*, p. 286–292.
- [BAC 94] BACCHUS F. et YANG Q., Downward refinement and the efficiency of hierarchical problem solving. *Artificial Intelligence*, vol. 71, 1994, p. 43–100.
- [BÄC 95] BÄCKSTRÖM C. et JONSSON P., Planning with Abstraction Hierarchies can be Exponentially Less Efficient. In : *Proceedings IJCAI-95*, p. 1599–1604.
- [BAR 92] BARRETT A. et WELD D., *Evaluating possible efficiency gains*, technical note, University of Washington, Department of Computer Science and Engineering, 1992.
- [FIN 93] FINK E. et YANG Q., Forbidding Preconditions and Ordered Abstraction Hierarchies. In : *Proceedings Spring Symposium AAAI*.
- [FIN 95] FINK E. et YANG Q., Planning with Primary Effects: Experiments and Analysis. In : *Proceedings IJCAI-95*, p. 1606–1611.
- [GAR 96] GARCIA F. et LABORIE P., Hierarchisation of the search space in temporal planning. In : *New Directions in AI Planning*, M. Ghallab and A. Milani (Eds.), IOS Press.
- [GHA 83] GHALLAB M. et ALLARD D., A_ε : an efficient near admissible heuristic search algorithm. In : *Proceedings IJCAI-83*.
- [GHA 89] GHALLAB M. et MOUNIR-ALAOUI A., Managing Efficiently Temporal Relations Through Indexed Spanning Trees. In : *Proceedings IJCAI-89*.

- [GHA 94] GHALLAB M. et LARUELLE H., Representation and Control in Ixtet, a Temporal Planner. In : *Proceedings AIPS-94*, p. 61–67.
- [KNO 90] KNOBLOCK C., Learning abstraction hierarchies for problem solving. In : *Proceedings AAAI-90*, p. 923–928.
- [KNO 91a] KNOBLOCK C., Search Reduction in Hierarchical Problem Solving. In : *Proceedings AAAI-91*, p. 686–691.
- [KNO 91b] KNOBLOCK C., TENENBERG J. et YANG Q., Characterizing Abstraction Hierarchies for Planning. In : *Proceedings AAAI-91*, p. 692–697.
- [KNO 94] KNOBLOCK C., Automatically generating abstractions for planning. *Artificial Intelligence*, vol. 68, 1994, p. 243–302.
- [LAB 95a] LABORIE P., *IxTeT : une Approche Intégrée pour la Gestion de Ressources et la Synthèse de Plans*, Thèse de doctorat, École Nationale Supérieure des Télécommunications, Paris, décembre 1995.
- [LAB 95b] LABORIE P. et GHALLAB M., Planning with Sharable Resource Constraints. In : *Proceedings IJCAI-95*, p. 1643–1649.
- [LAR 94] LARUELLE H., *Planification Temporelle et Exécution de Tâches en Robotique*, Toulouse, Thèse de doctorat, Université Paul Sabatier, Avril 1994.
- [MCA 91] MCALLESTER D. et ROSENBLITT D., Systematic nonlinear planning. In : *Proceedings AAAI-91*, p. 634–639.
- [MIS 91] MISSIAEN L., *Localized Abductive Planning with the Event Calculus*, Thèse de doctorat, Dept of Computer Science, K.U.Leuven, 1991.
- [PEN 92] PENBERTHY J. et WELD D., UCPop: A Sound, Complete, Partial Order Planner for ADL. In : *Proceedings 3rd International Conference on Knowledge Representation and Reasoning, Cambridge, MA*, p. 103–114.
- [PEN 94] PENBERTHY J. et WELD D., Temporal planning with continual change. In : *Proceedings AAAI-94*.
- [PUR 83] PURDOM P., Search rearrangement backtracking and polynomial average time. *Artificial Intelligence*, vol. 21, n° 1,2, 1983, p. 117–133.
- [SAC 74] SACERDOTI E., Planning in a Hierarchy of Abstraction Spaces. *Artificial Intelligence*, vol. 5, 1974, p. 115–135.
- [WEL 95] WELD D., An introduction to least commitment planning. *AI Magazine*, 1995, p. 27–61.
- [WIL 88] WILKINS D., *Practical Planning*, San-Mateo, CA, Morgan Kaufmann, 1988.

[YAN 90] YANG Q. et TENENBERG J., ABTWEAK: Abstracting a Nonlinear Least Commitment Planner. In : *Proceedings AAAI-90*, p. 204–209.

A Annexe

A.1 Preuves des énoncés

Théorème 1 : *Les hiérarchies d'abstraction utilisées dans IXET vérifient la propriété de monotonie ordonnée : pour chaque plan partiel courant possible, aucun affinement envisageable de ce plan partiel suite à la résolution de défauts ne fera apparaître de nouveau défaut qui soit d'un niveau strictement supérieur au niveau d'abstraction courant.*

Pour prouver cet énoncé, nous montrons tout d'abord les deux lemmes suivants :

Lemme 1 : *Soient $(R_0, C_0), \dots, (R_n, C_n)$ une séquence d'états d'abstraction obtenus par mises à jours successives. Alors $\forall att_n \in C_n$, il existe une séquence de noms d'attributs att_0, \dots, att_n , avec $\forall i att_i \in C_i$ et $att_{i+1} = att_i$ ou $[att_{i+1}] < [att_i]$.*

Preuve : Immédiat par récurrence sur la règle de mise à jour. Si $att_n \in C_n$ c'est soit qu'il était dans C_{n-1} et qu'il n'a pas été encore résolu : on pose $att_{n-1} = att_n$; soit qu'il vient d'être ajouté comme un $att_{nouveau}$, et on pose $att_{n-1} = att_{résolu}$, avec ainsi $[att_n] < [att_{n-1}]$. \square

Lemme 2 : *Pour tout état d'abstraction (R, C) on a $R \cap C = \emptyset$. De plus, si att et att' sont deux noms d'attributs de C , alors soit $[att] = [att']$, soit $[att]$ et $[att']$ sont non comparables, c'est-à-dire que l'on ne peut avoir $[att] < [att']$ ni $[att'] < [att]$.*

Preuve : Par récurrence sur la taille de toute séquence d'états d'abstraction $(R_0, C_0), \dots, (R_n, C_n)$ le long d'un chemin de l'arbre de recherche. Pour $n = 0$, les deux propriétés sont vérifiées du fait de la définition de R_0 et C_0 . Soit $n > 0$, et supposons les propriétés vraies jusqu'à $n - 1$.

$$\begin{aligned} C_n \cap R_n &= (C_{n-1} - [att_{résolu}] \cup \{att_{nouveau}\}) \cap (R_{n-1} \cup [att_{résolu}]) \\ &= \{att_{nouveau}\} \cap (R_{n-1} \cup [att_{résolu}]) \\ &= (\{att_{nouveau}\} \cap R_{n-1}) \cup (\{att_{nouveau}\} \cap [att_{résolu}]) \end{aligned}$$

On a $\{att_{nouveau}\} \cap [att_{résolu}] = \emptyset$ car $[att_{nouveau}] < [att_{résolu}]$. Supposons alors qu'il existe $att_{nouveau} \in R_{n-1}$. Dans la séquence d'états d'abstraction soit (R^-, C^-) le dernier état tel que $att_{nouveau} \notin R^-$ et $att_{nouveau} \in C^-$. D'après le lemme 1, pour $att_{nouveau} \in C_n$, $\exists att \in C^- \mid [att_{nouveau}] < [att]$. Puisque att et $att_{nouveau}$ sont dans C^- , il y a contradiction avec l'hypothèse de récurrence et donc $C_n \cap R_n = \emptyset$.

Supposons maintenant qu'il existe att, att' dans C_n comparables entre eux. Puisque par récurrence att et att' ne peuvent être tous deux dans C_{n-1} , la seule possibilité est que $\exists att \in C_{n-1}, att \notin [att_{résolu}]$ et $att_{nouveau} \in C_n$ tels que $[att] < [att_{nouveau}]$ ou $[att_{nouveau}] < [att]$. Comme $[att_{nouveau}] < [att_{résolu}]$, $[att] < [att_{nouveau}] \Rightarrow [att] < [att_{résolu}]$, ce qui est impossible dans C_{n-1} par hypothèse de récurrence. Par la définition 9 de $att_{nouveau}$, si $[att_{nouveau}] < [att]$, alors $att \in R_n$. Puisque $att \notin [att_{résolu}]$, cela implique que $att \in R_{n-1}$, ce qui est impossible car $C_{n-1} \cap R_{n-1} = \emptyset$ par récurrence. Ainsi, $\forall att, att' \in C_n, [att] \not< [att']$. \square

Preuve du théorème 1 : Soit (R, C) l'état d'abstraction courant, et L le niveau d'abstraction associé. Soit Φ un défaut de L ($att_\Phi \in C$). Supposons que le traitement de Φ conduise à l'apparition d'un nouveau défaut Φ' . Nous devons montrer que nécessairement $att_{\Phi'} \notin R$.

Supposons que $att_{\Phi'} \in R$. Parmi les différentes résolvantes d'un défaut, la seule manière d'obtenir un nouveau défaut sur $att_{\Phi'}$ est d'insérer une tâche \mathcal{T} pour résoudre Φ (en effet, l'ajout de contraintes de précédence temporelle et de contraintes sur les variables ne peut que réduire l'ensemble global des défauts du plan, et l'ajout d'un lien causal ne peut créer de nouvelles menaces que sur un nom d'attribut de C). D'après la définition 6 des contraintes \prec sur les noms d'attributs, l'analyse de la tâche \mathcal{T} apportera nécessairement la contrainte $att_{\Phi'} \prec att_\Phi$, soit $[att_{\Phi'}] < [att_\Phi]$ ou $[att_{\Phi'}] = [att_\Phi]$.

Le long du chemin arrivant jusqu'au nœud de recherche courant, soit (R^-, C^-) le dernier état d'abstraction tel que $att_{\Phi'} \notin R^-$ et $att_{\Phi'} \in C^-$. A partir de $[att_{\Phi'}] < [att_\Phi]$ ou $[att_{\Phi'}] = [att_\Phi]$ et du lemme 2, on en déduit que $att_\Phi \notin C^-$. Alors en appliquant le lemme 1, $\exists att \in C^- [att_\Phi] < [att]$, ce qui entraîne $[att_{\Phi'}] < [att]$, et contredit le lemme 2 puisque $att_{\Phi'}$ et att appartiennent au même ensemble C^- . \square

Théorème 2 : Le nombre maximal l des différents niveaux d'abstraction parcourus au cours de la recherche depuis le scénario initial \mathcal{P}_0 jusqu'à un plan solution \mathcal{P} est tel que $d_G \leq l \leq n_G$.

Preuve : Le terme *maximal* signifie ici que les opérations de mise à jour de l'état d'abstraction (R, C) sont poursuivies de $C = C_0$ jusqu'à $C = \emptyset$. Il est alors clair que l est au moins aussi grand que la longueur de tout chemin reliant un élément maximal (sans père) à un élément minimal (sans fils) de \mathcal{G} , et donc que $d_G \leq l$. Inversement, à chaque mise à jour de l'état d'abstraction, l'ensemble R est augmenté d'exactement un nœud de \mathcal{G} , et le théorème 1 montre que chaque classe ne peut ainsi être ajoutée à R qu'au plus une seule fois. On en déduit donc que $l \leq n_G$. \square

Théorème 3 : En l'absence de prédictats de type produce,
 $\forall \mathcal{P} \in COMPL(\mathcal{P}_0), \forall \Phi, \Phi'$ défauts de \mathcal{P} ,

$$att_\Phi \prec att_{\Phi'} \Rightarrow att_\Phi \prec_{SI} att_{\Phi'}$$

Preuve : Nous montrons tout d'abord que si la définition 6 conduit à poser la contrainte $att_{\Phi} \prec att_{\Phi'}$, alors la définition 11 amène aussi à poser $att_{\Phi} \prec_{SI} att_{\Phi'}$. En effet, pour pouvoir ajouter la contrainte $att_{\Phi} \prec att_{\Phi'}$, il est nécessaire que $att_{\Phi'}$ soit utilisé dans au moins un événement d'une tâche \mathcal{T} du problème qui contient aussi une proposition temporelle sur l'attribut att_{Φ} . Il est clair que comme Φ' est un défaut d'un plan partiel issu de \mathcal{P}_0 , $att_{\Phi'}$ est pertinent pour le scénario initial. D'autre part, comme les événements expriment à la fois une condition et un effet sur l'attribut, il est aisément de voir que dès qu'un attribut associé à un événement est pertinent dans une tâche, il est aussi pertinent dans toutes les tâches qui contiennent un événement sur ce même attribut. Ceci permet de voir que $att_{\Phi'}$ est un attribut pertinent dans \mathcal{T} pour \mathcal{P}_0 , et donc, d'après la définition 11, que $att_{\Phi} \prec_{SI} att_{\Phi'}$.

On montre alors la propriété. Si on a $att_{\Phi} \prec att_{\Phi'}$, c'est qu'il existe un chemin $att_{\Phi} = att_0 \prec att_1 \prec \dots \prec att_{n-1} \prec att_n = att_{\Phi'}$ où les contraintes $att_i \prec att_{i+1}$ sont posées par l'analyse des tâches du problème. Sur ce chemin, chacun des attributs att_i , $i > 0$, intervient nécessairement dans un événement d'une tâche. On peut alors facilement se convaincre dans ce cas que chacun de ces attributs att_i est pertinent pour \mathcal{P}_0 , et que donc on a aussi $att_{\Phi} = att_0 \prec_{SI} att_1 \prec_{SI} \dots \prec_{SI} att_{n-1} \prec_{SI} att_n = att_{\Phi'}$, soit $att_{\Phi} \prec_{SI} att_{\Phi'}$. \square

A.2 Application COLUMBUS

Ce domaine a été modélisé dans le cadre du projet PADRE du MESR en collaboration avec les sociétés IXI et Matra Marconi Space. Il s'agit de planifier des expériences de biologie à bord d'une station spatiale de type Columbus. La figure 7 schématisé l'environnement du laboratoire. Les expériences à planifier consistent à manipuler des cultures en utilisant certains équipements scientifiques (incubateur, centrifugeuse, microscope, ...). Deux robots peuvent se déplacer dans le laboratoire et déplacer les objets. La plupart des expériences proprement dites se font au moyen d'un bras manipulateur.

Cet environnement fait intervenir un certain nombre de ressources non-partageables (les instruments scientifiques, les robots, le bras manipulateur, les différents éléments manipulés), mais aussi des ressources partageables (la puissance électrique instantanée maximale, le débit des canaux de communication).

D'autre part, il s'agit d'un problème de synthèse de plans dans la mesure où nous spécifions uniquement l'état final des cultures sur lesquelles doivent porter les expériences.

La description du domaine dans le formalisme IXTE (constantes du domaine, attributs d'état et de ressources, modèles de tâches) est donnée ci-dessous. La hiérarchie générée automatiquement à partir de cette modélisation est celle que nous avons déjà vue sur la figure 4.

A.2.1 Constantes du domaine

```
constant ROBOTS = { Rb1, Rb2 };
constant ZONES = { Banc, Stock, Centrif, Poubelle, Incub, Micro, Mouvmt } | ROBOTS;
constant CULTURES = { Culture1, Culture2 };
constant TUBES = { Tube1, Tube2 };
```

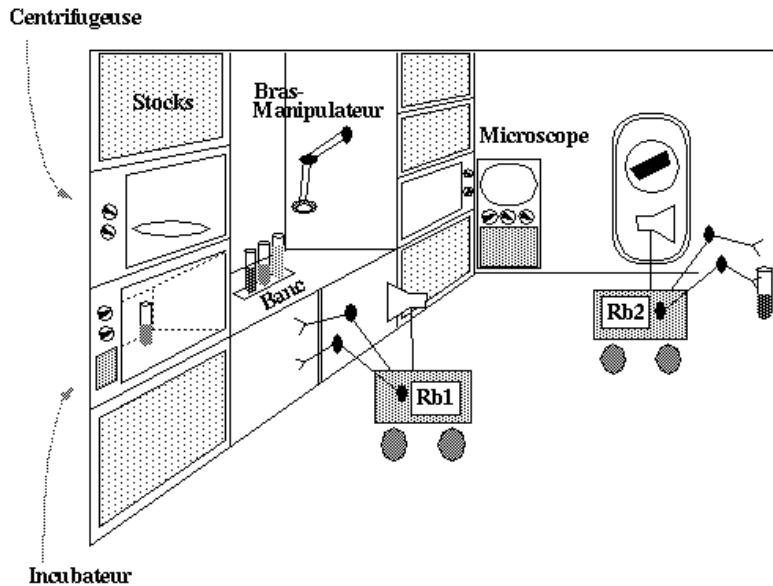


Figure 7. Le domaine COLUMBUS

```

constant FILTRES = { Filtre1, Filtre2, Filtre3 };
constant FLASQUES = { Flasque1, Flasque2 };
constant SERINGUES = { S1, S2, S3, S4, S5 };
constant ELEMENTS = CULTURES | FILTRES | FLASQUES | SERINGUES | TUBES;
constant VITESSES = {V0, V1, V2};
constant TEMPERATURES = { Amb, T20, T30 };
constant BOOLEEN = { vrai, faux };

```

A.2.2 Attributs du domaine

```

attribute POSITION(?x) {
    ?x in ELEMENTS;
    ?value in ZONES;
}
attribute LOCALISATION(?x) {
    ?x in ROBOTS;
    ?value in ZONES;
}
attribute TEMPERATURE() {
    ?value in TEMPERATURES;
}
attribute VITESSE() {
    ?value in VITESSES;
}
attribute INCUBE(?Elt,?Temp) {
    ?Temp in TEMPERATURES;
    ?Elt in CULTURES;
    ?value in BOOLEEN;
}
attribute CENTRIFUGE(?Elt,?Vits) {
    ?Vits in VITESSE;
    ?Elt in CULTURES;
    ?value in BOOLEEN;
}
attribute DISSOCIE(?Elt) {
    ?Elt in CULTURES;
    ?value in BOOLEEN;
}
attribute FIXE(?Elt) {
    ?Elt in CULTURES;
}

```

```
?value in BOOLEEN;}
```

A.2.3 Ressources du domaine

```
resource ROBOT(?r) { ?r in ROBOTS;
    capacity = 1; }
resource BRAS_MANIPULATEUR { capacity = 1; }
resource MICROSCOPE() { capacity = 1; }
resource CENTRIFUGEUSE() { capacity = 1; }
resource PUISSANCE() { capacity = 100; }
resource COMMUNICATION() { capacity = 800; }
resource BANC_TRAVAIL() { capacity = 2; }
resource INCUBATEUR() { capacity = 5; }
resource ZONE_DEPLACEMENTO {
    capacity = 3; }
```

A.2.4 Description des tâches :

```
task POMPER(?Elt,?S)(start, end) {
    ?S in SERINGUES;
    hold (POSITION(?Elt):Banc, (start,end));
    use (PUISANCE:10, (start, end));
    use (COMMUNICATION:30, (start, end));
    use (BRAS_MANIPULATEUR:1, (start, end));
    (end - start) in [00:01:00, 00:03:00]; }

task INJECTER(?Elt,?S)(start, end) {
    ?S in SERINGUES;
    hold (POSITION(?Elt):Banc, (start,end));
    use (PUISANCE:10, (start, end));
    use (COMMUNICATION:30, (start, end));
    use (BRAS_MANIPULATEUR:1, (start, end));
    (end - start) in [00:01:00, 00:03:00]; }

task CHAUFFER(?Temp)(start, end) {
    event (TEMPERATURE):(Amb, ?Temp), start);
    event (TEMPERATURE):(?Temp, Amb), end);
    hold (TEMPERATURE():?Temp, (start,end));
    use (PUISANCE:40, (start, end)); }

task OBSERVER(?Elt)(start, end) {
    hold (POSITION(?Elt):Micro, (start,end));
    use (PUISANCE:10, (start, end));
    use (COMMUNICATION:400, (start, end));
    use (MICROSCOPE:1, (start, end));
    (end - start) in [00:15:00, 00:20:00]; }

task INCUBER(?Elt, ?Temp)(start, end){
    hold (TEMPERATURE():?Temp, (start, end));
    hold (POSITION(?Elt):Incub, (start, end));
    event (INCUBE(?Elt,?Temp):(faux, vrai), end);
    use (PUISANCE:8, (start, end));
    use (COMMUNICATION:50, (start, end));
    use (INCUBATEUR:1, (start, end));
    (end - start) in [00:09:00, 00:10:00];}

task CENTRIFUGER(?Elt, ?Vits)(start, end){
    hold (POSITION(?Elt):Centrif, (start, end));
    event (VITESSE():(V0, ?Vits), start);
    event (VITESSE()):(?Vits, V0), end);
    hold (VITESSE():?Vits, (start, end));
    event (CENTRIFUGE(?Elt,?Vits):(faux, vrai), end);
    use (PUISANCE:20, (start, end));
    use (COMMUNICATION:40, (start, end));
    use (CENTRIFUGEUSE:1, (start, end));
    (end - start) in [00:09:00, 00:10:00];}
```

```

task FIXATION(?Elt,?S,?T)(start, end){
    ?S in SERINGUES;
    ?T in TUBES;
    hold (POSITION(?Elt):Banc, (start, end));
    hold (POSITION(?T):Banc, (start, end));
    event (FIXE(?Elt):(faux, vrai), end);
    use (BRAS_MANIPULATEUR:1, (start, end));
    use (PUISANCE:10, (start, end));
    use (COMMUNICATION:30, (start, end));
    (end - start) in [00:06:00, 00:08:00]; }

task DISSOCIATION(?Elt,?S,?F)(start, end){
    ?S in SERINGUES;
    ?F in FILTRES;
    hold (POSITION(?Elt):Banc, (start, end));
    hold (POSITION(?F):Banc, (start, end));
    event (DISSOCIE(?Elt):(faux, vrai), end);
    use (BRAS_MANIPULATEUR:1, (start, end));
    use (PUISANCE:8, (start, end));
    use (COMMUNICATION:20, (start, end));
    (end - start) in [00:05:00, 00:07:00]; }

task ALLER_A(?r,?Zone1,?Zone2)(start, end){
    event (LOCALISATION(?r):(?Zone1, Mouvmt), start);
    event (LOCALISATION(?r):(Mouvmt,?Zone2) , end);
    hold (LOCALISATION(?r):Mouvmt, (start, end));
    use (ROBOT(?r):1, (start, end));
    use (PUISANCE:30, (start, end));
    use (COMMUNICATION:300, (start, end));
    use (ZONE_DEPLACEMENT:1, (start, end));
    (end - start) in [00:02:00, 00:04:00]; }

task INSTALLER(?r,?Elt,?Zone)(start, end){
    variable ?Zone0,?ZoneR;
    event (POSITION(?Elt):(?Zone0,?r), start);
    event (POSITION(?Elt):(?r,?Zone) , end);
    hold (POSITION(?Elt):?r (start, end));
    event (LOCALISATION(?r):(?ZoneR, Mouvmt), start);
    event (LOCALISATION(?r):(Mouvmt,?Zone) , end);
    hold (LOCALISATION(?r):Mouvmt, (start, end));
    use (ROBOT(?r):1, (start, end));
    use (PUISANCE:40, (start, end));
    use (COMMUNICATION:400, (start, end));
    use (ZONE_DEPLACEMENT:1, (start, end));
    (end - start) in [00:03:00, 00:10:00]; }

task ELIMINER(?r,?Elt)(start, end){
    variable ?Zone0,?ZoneR;
    event (POSITION(?Elt):(?Zone0,?r), start);
    event (POSITION(?Elt):(?r, Poubelle) , end);
    hold (POSITION(?Elt):?r, (start, end));
    event (LOCALISATION(?r):(?ZoneR, Mouvmt), start);
    event (LOCALISATION(?r):(Mouvmt,?Zone) , end);
    hold (LOCALISATION(?r):Mouvmt, (start, end));
    use (ROBOT(?r):1, (start, end));
    use (PUISANCE:40, (start, end));
    use (COMMUNICATION:400, (start, end));
    use (ZONE_DEPLACEMENT:1, (start, end));
    (end - start) in [00:03:00, 00:10:00]; }

```