Département Informatique

Diplôme préparé : DUT informatique

Conception d'un module de détection d'anomalie dans l'exécution d'un plan

Ombredane César

TUTEUR ENSEIGNANT

JEAN MASSARDI

RESPONSABLE EN ENTREPRISE

ÉRIC BEAUDRY

ANNEE UNIVERSITAIRE 2019/2020

STAGE DU 27/04 AU 24/07/20

SOUTENANCE DU 30/06/20

REMERCIEMENTS

Je souhaite d'abord remercier Mme Cécile Balkanski pour m'avoir donné l'opportunité de réaliser ce stage ainsi que pour toute l'aide qu'elle m'a apporté tout au long de ce dernier.

Je souhaite également remercier M. Éric Beaudry qui m'a accueilli au sein de son laboratoire et qui, tout le long de mon stage, m'a suivi et aidé dans la réalisation de mes missions.

Enfin, Je remercie mon tuteur durant ce stage, M. Jean Massardi pour son encadrement et son aide ainsi que pour sa compréhension des difficultés rencontrées.

Résumé

Dans le cadre de ma 2eme année de DUT Informatique à l'IUT d'Orsay et de ma mobilité internationale à l'UQAM, j'ai réalisé un stage de recherche de 13 semaines dans la période du 27/04/20 au 24/07/20.

Cette expérience m'a donné l'occasion de travailler pour Age-Well dans le cadre du projet Vigil dirigé par Éric Beaudry. Age-Well est un réseau de centres d'excellence du Canada portant sur les technologies liées aux séniors et aux aidants.

Durant ce stage, je me suis penché sur l'algorithme PARC qui est un algorithme de reconnaissance de plan. Mes missions ont été dans un premier temps de nettoyer le code de PARC puis de développer un algorithme de détection d'anomalies dans l'exécution d'un plan.

Etant donné la crise sanitaire mondiale actuelle, je n'ai pas pu rester au Canada. J'ai donc réalisé la totalité de mes missions depuis la France.

J'ai donc commencé par nettoyer complètement le code de l'algorithme PARC ce qui m'a permis de le comprendre et de me familiariser avec le projet. J'ai ensuite réfléchi et développé un algorithme de détection d'anomalies dans l'exécution d'un plan. Tout mon code est basé sur les recherches de mon maître de stage Jean Massardi. Enfin, j'ai réalisé de nombreux tests sur mon algorithme pour vérifier et quantifier son efficacité.

Mon stage se terminant le 24 Juillet, il me restera encore du travail après ma soutenance. Mes objectifs sont premièrement d'implémenter la détection de comportements anormaux au projet Vigil dans son ensemble et pouvoir réaliser des tests pratiques et non théoriques. Et enfin, s'il me reste du temps, j'aimerais développer un second algorithme visant à dynamiser un paramètre de la détection de comportements anormaux.

Ce stage est pour moi une expérience très enrichissante tant sur le plan technique car j'y apprends de nouvelles choses que sur le plan professionnel.

Table des matières

REMERCIEMENTS **3**
Résumé **4**
Table des matières **5**
Introduction **6**
Présentation du projet **8**
Projet Vigil 8
Age-Well 8
Laboratoire 8
PARC **9**
Bibliothèque de plan 10
Filtre à particules 12
Implémentation 14
Expansion de l'arbre d'une particule 16
Initialisation 17
Filtrage 18
Résumé 18
Cognitive Distress Management (CDM) **19**
Méthodes 20
Sum 21
Min 21
Support 21
Tests 22
Implémentation 23
Résultats 24
Sum 24
Min 25
Support 26
Résumé 27
Bilan **28**
Lexique **29**
Table des illustrations **30**
Annexes **31**

Introduction

Pour conclure mon DUT Informatique à l'IUT d'Orsay, j'ai pu réaliser un stage en laboratoire de 13 semaines du 27/04 au 24/07/2020. Ce stage a pour but de valider les compétences acquises au cours de mon DUT ainsi que de m'introduire au monde du travail et aux contraintes de ce dernier.

J'ai eu l'opportunité de réaliser mon stage au sein des laboratoires de l'UQAM dans le cadre de ma mobilité internationale. J'ai été assigné au projet Vigil pour le centre Age-Well. Ce stage m'a permis de comprendre le fonctionnement du système de publication scientifique, et, plus largement, de la recherche en informatique.

Suite à la crise actuelle de COVID-19 et à mon rapatriement en France, j'ai dû faire la totalité de mon travail au sein de ce projet en télétravail et avec un décalage horaire de 6h par rapport à mon maître de stage et aux autres collaborateurs. Cela n'a pas été de tout repos mais m'a permis de me rendre compte de la difficulté de travailler seul et a grandement participé à améliorer mon autonomie.

Durant ce stage j'ai donc pu m'intégrer à un projet de recherche. Mes missions consistaient à, dans un premier temps, me familiariser avec le projet et apprendre les technologies nécessaires à son développement, puis, à développer un module de détection d'anomalie dans l'exécution d'un plan.

Ma soutenance de stage ayant lieu avant la fin de celui-ci, il me reste encore des missions à accomplir. Mais pour le moment, les enjeux de ce stage ont été de m'adapter à un projet dans un délai très court et apprendre des technologies dans ce même délai. De plus, cela m'a permis de développer mes capacités en algorithmie car ce projet est bien plus complexe que tous ceux auxquels j'ai pu participer jusque-là.

Dans le cadre de cette recherche, j'ai été affecté au développement de l'Algorithme PARC qui est un algorithme de reconnaissance de plan. Cet algorithme est une des deux parties principales du projet Vigil et se décompose en trois parties : la définition des bibliothèques de plan, l'algorithme de reconnaissance de plan (déjà fonctionnel lors de mon arrivé sur le projet) et l'algorithme de détection de comportements anormaux.

Mon objectif, après m'être approprié le projet, a été de développer l'algorithme de détection de comportements anormaux en me basant sur les formules mathématiques déjà fiables données par Jean Massardi, mon maître de stage.

Dans ce rapport, je présenterai d'abord le projet Vigil et expliquerai l'algorithme PARC. Puis, je présenterai mon travail et les résultats que cela a apporté. Enfin, je ferai le bilan de mon stage jusque-là et présenterai mes objectifs pour la suite de ce dernier.

Présentation du projet

Projet Vigil

Le projet Vigil a pour objectif de développer un robot mobile pour aider les personnes âgées qui vivent encore chez elles. Le but de cette technologie est de pouvoir détecter les activités de son utilisateur et de pouvoir l'assister dans des tâches simples telles que faire à manger, faire de l'exercice ou autre.

Dans ce projet très large, j'ai été affecté au développement de l'application de détection de plan. Il consiste dans un premier temps à détecter les actions d'une personne avec une unique caméra mobile puis, dans un deuxième temps, en fonction de ces actions, reconnaitre le plan de l'utilisateur afin de l'aider dans ce plan. Ou, si l'utilisateur dévie de ce dernier, lui rappeler le plan courant.

Age-Well

Ce projet fait partie des projets de recherche principaux de l'institut Age-Well. Age-Well est un centre de recherche d'excellence ayant pour but de développer des technologies pour aider les personnes âgées et/ou avec une déficience cognitive. Ce centre travaille avec de nombreuses universités dont l'UQAM où j'ai fait ma mobilité internationale.

Laboratoire

Le laboratoire dans lequel j'ai travaillé est le laboratoire CRIA de la faculté des sciences à l'UQAM avec pour responsable Éric Beaudry.

Laboratoire CRIA : http://gdac.uqam.ca/CRIA/

La faculté des sciences est située dans le Complexe des Pierre-Dansereau.

Figure 1 : Le pavillon Président-Kennedy (PK)
de la faculté des sciences - UQAM

PARC

PARC est un algorithme de reconnaissance de plan. Cela veut dire qu'il doit reconnaitre, le plus vite possible, un plan appartenant à une bibliothèque en se basant sur des observations. Il est donc important pour le comprendre, d'expliquer le fonctionnement de la bibliothèque de plan utilisée.

Cet algorithme a aussi pour but de pouvoir être exécuté en temps réel. C'est un véritable défi car jusque-là, les algorithmes de reconnaissance de plan étaient plutôt lents. Cela vient directement de la méthode utilisée.

Voici l'ancienne méthode de reconnaissance de plan :

Tous les plans sont stockés dans une grande liste. Un plan étant composé d'une séquence d'actions atomiques (non décomposables) et du but que ce plan vise à atteindre. À chaque nouvelle observation, tous les plans ne correspondant pas sont écartés. Cette méthode est très simple mais présente beaucoup de défauts. Premièrement, elle est très lente car dans une bibliothèque de plan il y a rarement une seule façon d'atteindre un objectif. Cela fait donc énormément de plans et donc plus de temps de calcul. Deuxièmement, cette méthode n'a aucune résistance au bruit. Ici le bruit est défini comme une erreur d'observation. Si cela survient, la détection de plan sera à coup sûr faussée.

C'est la raison pour laquelle pour PARC, nous avons mis en place un algorithme de reconnaissance de plan basé sur un filtre à particule qui vous sera expliqué dans un deuxième temps. Mais il est d'abord nécessaire de définir certains termes :

Plan : un plan est un ensemble d'actions partiellement ordonnées, qui vise à atteindre un objectif.

Action : une action est un élément de la bibliothèque de plan. On dit qu'une action est atomique (ou Terminal) quand elle n'est pas décomposable. Elle est ainsi définie comme une observation.

Objectif : un objectif est la finalité d'un plan considéré comme un but. Exemple : *Faire bouillir de l'eau* est la finalité d'un plan mais pas un but en soi alors que *Faire du thé* est la finalité d'un plan et un objectif.

Bibliothèque de plan

Pour que le système de reconnaissance de plan fonctionne de manière optimale, il a fallu revoir le système de stockage de la bibliothèque de plan.

Une bibliothèque de plan contient :

- Une liste A des actions atomiques (non décomposables) définie comme les actions observables par l'algorithme de reconnaissance d'activité,
- Une liste NT des actions non atomiques (décomposables),
- Une liste G des objectifs avec $G \in NT$. Un objectif est défini comme une finalité (préparer du thé est une finalité là où faire bouillir de l'eau n'en est pas une),
- Une liste P de règles de production de la forme $\alpha \rightarrow S, \sigma$ avec α une action non atomique ($\alpha \in NT$), S un ensemble d'actions ($S \in A \cup NT$) et σ un ensemble de règles de production.

De cette manière, il n'y a qu'un plan pour chaque action non Atomique. Ce qui résout le problème des bibliothèques de plan trop grandes.

Prenons un exemple de bibliothèque de plan simple :

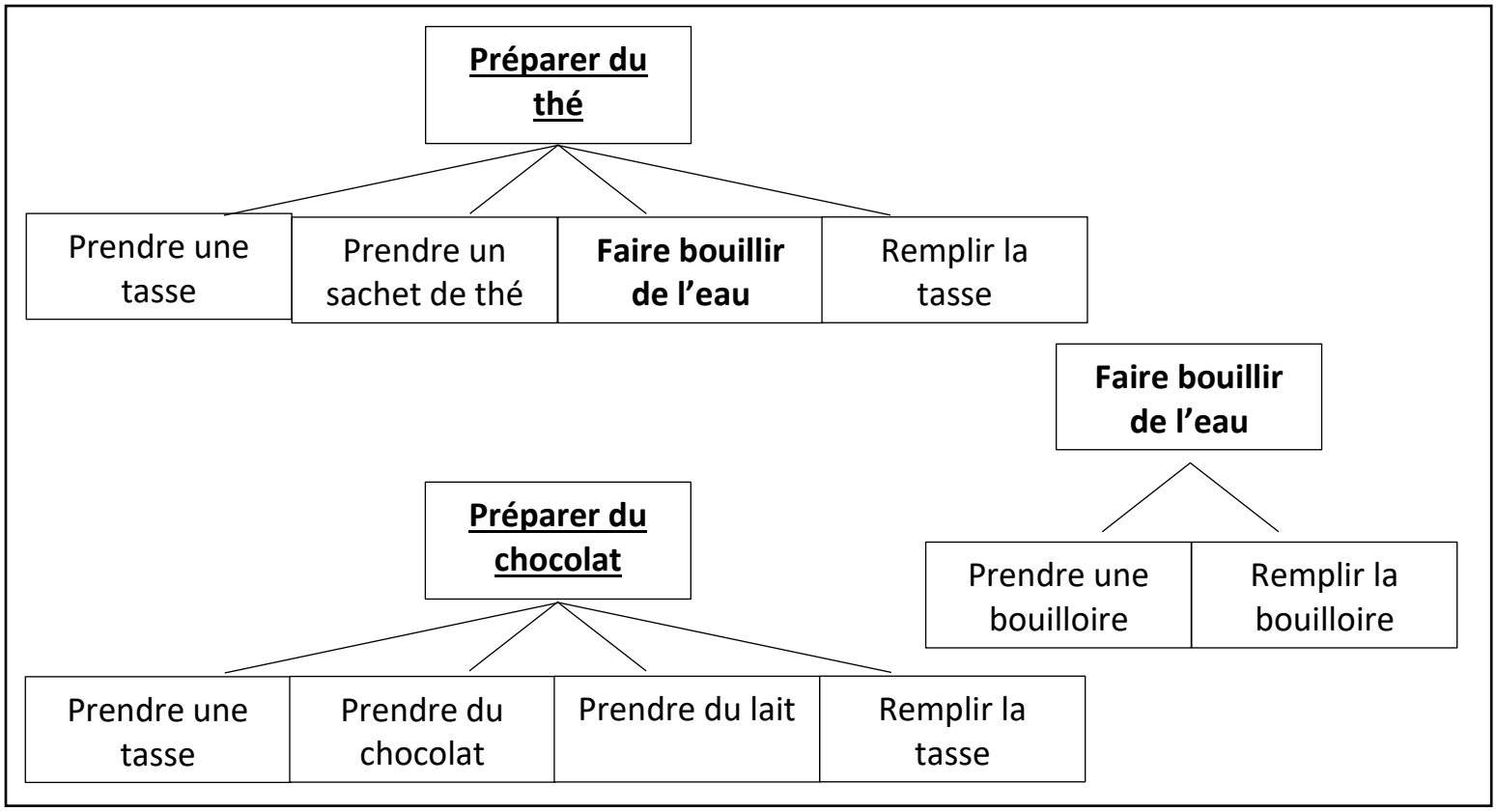


Figure 2 : représentation d'une bibliothèque de plan sous forme d'arbres

Ici nous avons trois actions non atomiques : **Faire bouillir de l'eau**. Dont deux objectifs : **préparer du thé** et **préparer du chocolat**. Les actions restantes sont atomiques. La bibliothèque de plan sera donc la suivante :

A = {prendre une tasse, prendre un sachet de thé, remplir la tasse, prendre du chocolat, Prendre du lait, prendre une bouilloire, remplir la bouilloire}

NT = {préparer du thé, préparer du chocolat, faire bouillir de l'eau}

G = {préparer du thé, préparer du chocolat}

P = {
{faire bouillir de l'eau → prendre une bouilloire, remplir la bouilloire, σ = {(1,2)}},
{préparer du thé → prendre une tasse, prendre un sachet de thé, faire bouillir de l'eau, remplir la tasse, σ = {(1,4),(2,4),(3,4)}},
{préparer du chocolat → prendre une tasse prendre du chocolat, prendre du lait, remplir la tasse, σ = {(1,4),(2,4),(3,4)}}
}

Figure 3 : Bibliothèque de plan

Pour l'ordre partiel des plans, ce sont les indices des actions dans la liste précédente qui sont utilisés et non les actions elles-mêmes. C'est seulement pour pouvoir y accéder plus rapidement.

Dans la bibliothèque de plan, nous stockons aussi d'autres informations :

- Premièrement, pour l'optimisation du système :

Dans la bibliothèque de plan et dans le reste du module, toutes les actions sont représentées par des objets de type *Integer*. C'est simplement une question d'optimisation, les types *String* étant longs à comparer et beaucoup plus lourds que les types *Integer*, pour gagner du temps de calcul, il est nécessaire de représenter les actions par des objets légers. Il existe donc deux *maps* dans la bibliothèque de plan pour faire la conversion entre le nom d'une action (type *String*) et son numéro (type *Integer*).

- Deuxièmement, pour l'ajout d'informations utiles :

La première information utile pour la programmation de la reconnaissance de plan est la représentation du bruit. Cela se fait sous forme d'une fonction de distribution de probabilité : la probabilité de confondre une action avec une autre ou elle-même (dans le cas où l'observation est la bonne). Cette fonctionnalité nécessite une variable représentant la prévision du bruit.

La deuxième information est elle aussi une fonction de distribution de probabilité mais cette fois-ci pour modéliser la probabilité d'utiliser un certain plan pour une action. En effet, avec l'utilisation de plans partiellement ordonnés, nous réduisons considérablement le nombre de plans différents pour une même action mais ne le réduisons pas à son minimum (un seul plan). Il reste donc plusieurs plans pour effectuer une même action et il est donc nécessaire de modéliser la probabilité d'utiliser ces plans.

Filtre à particule

Le système de reconnaissance de plan du module PARC utilise un filtre à particule. Mais dans ce cas précis, chaque particule représente en fait un plan sous forme d'un arbre. Pour comprendre cela, nous allons déjà nous attarder sur le fonctionnement du filtre à particule, puis sur le fonctionnement de chaque particule.

Un filtre à particule a un nombre de particules constant et représentatif de toute la bibliothèque de plan. Chaque particule représente un objectif (ici, faire du thé ou faire du chocolat) et une future action possible. Pour chaque observation possible, seules les particules qui la supportent resteront et les autres seront supprimées. Ensuite, nous procédons à un remplissage des particules qui supportent l'observation précédente. Les particules chercheront donc une autre action possible et ainsi de suite.

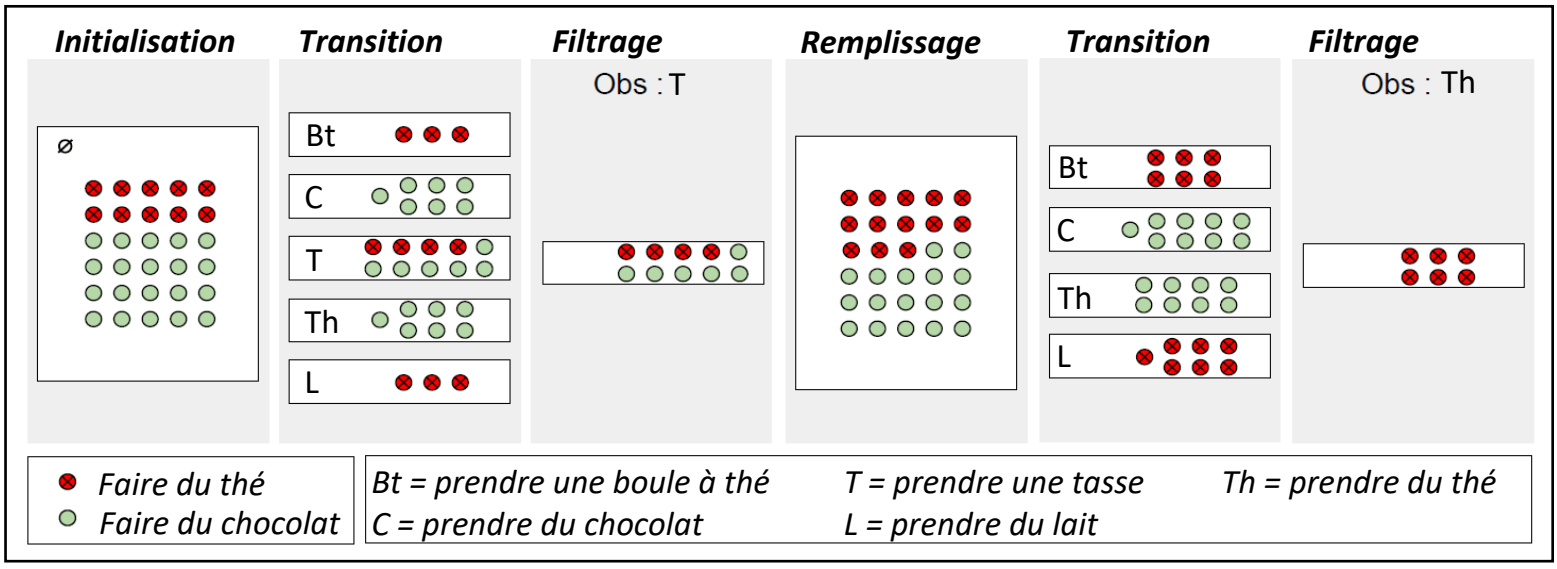


Figure 4 : représentation du filtre a particule

Dans la figure 4, lors de l'initialisation, une particule sur trois représente l'objectif *Faire du thé* alors que les autres représentent *Faire du chocolat*. Dans la phase de transition, nous voyons les observations prédites par chaque particule. Puis vient le filtrage : la détection d'activité transmet une observation et seules les particules qui supportent cette observation restent, les autres sont supprimées. Après cela, on complète avec de nouvelles particules qui supportent déjà l'observation précédente, jusqu'à atteindre le même nombre de particules qu'au début. Cette boucle se répète à chaque observation jusqu'à n'avoir plus que des particules qui représentent le même objectif. À ce moment, nous avons deviné l'objectif et donc le plan de l'utilisateur. Il est bien sûr aussi possible d'avoir une idée avant que toutes les particules représentent le même objectif en regardant simplement lesquels sont majoritaires.

Maintenant il est important d'expliquer comment chaque particule, indépendamment, prédit les actions suivantes. À l'initialisation, chaque particule se voit attribuer un objectif aléatoirement parmi ceux de la bibliothèque de plan (on fait bien sûr attention à ce que tous les objectifs soient représentés). Puis, en se servant de la répartition des plans, chaque particule se voit attribuer un plan aléatoire pour satisfaire son objectif (tous sont également représentés). Cette répartition prend en compte la probabilité de chaque plan défini dans la bibliothèque de plan. La répartition n'est donc pas égale. Puis, pour prédire la prochaine action possible, chaque particule choisit de manière aléatoire une action parmi celles possibles de son plan ou d'autres qui constituent des actions qui composent son plan. En représentant le plan sous forme d'arbre et en omettant les actions ayant des antécédents qui n'ont pas encore été exécutés, chaque particule présente un ordre d'action différent pour réaliser le plan. Tous les ordres possibles sont donc représentés si le nombre de particules est suffisant. Pour chaque observation, l'action transmise aux particules n'est pas toujours la même. En effet, si la prévision du bruit n'est pas égale à 0%, alors il y a une probabilité fixe pour que l'action observée ne soit pas l'action réelle. Dans ce cas, l'action transmise à chaque particule a une probabilité fixe d'être différente de l'action observée.

Cette méthode est bien plus rapide car le seul paramètre faisant varier la complexité en temps est le nombre de particules. Et selon les résultats des études réalisées par Jean Massardi, mon maître de stage, seules 100 particules sont nécessaires au bon fonctionnement de l'algorithme et 500 particules rendent son efficacité optimale.

Implémentation

Pour l'implémentation de l'algorithme de reconnaissance de plan et du système PARC en entier, nous utilisons le langage C++ car le principal défi est le temps de calcul et C++ permet de faire beaucoup de calculs en très peu de temps.

L'implémentation de la bibliothèque de plan se divise en quatre classes :

- Rule

Rule est une classe qui définit les règles de production d'une action (non Atomique). Elle contient un numéro pour l'identifier, l'action qu'elle doit réaliser (primitive), la liste d'actions qui la compose et une liste de doublons qui représentent deux indices de la liste précédente et qui signifie que la première action doit impérativement être effectuée avant la seconde.

- PlanLibrary

PlanLibrary est une bibliothèque de plan comme décrite dans la partie sur la bibliothèque de plan. Avec trois listes non ordonnées (représentant les actions atomiques, non atomiques et les objectifs) et une map qui contient les règles de production qui ont pour clé leur action primitive.

- ProbabilityDistribution

ProbabilityDistribution est simplement une distribution de probabilités ; elle est représentée comme suivant : $map < int, map < int, float >>$. Cette classe est utilisée pour stocker la probabilité d'utiliser un plan pour une action précise ou pour stocker la probabilité de confondre une action avec une autre (bruit). La somme des *floats* contenue dans la deuxième *map* est donc égale à 1.

- ExtendedPlanLibrary (EPL)

EPL est l'objet qui sera utilisé dans tout le reste du programme. Il contient une référence (pointeur) à un objet de type PlanLibrary, mais aussi deux *maps* qui permettent de faire la conversion entre le nom d'une action (*String*) et le numéro qui lui est attribué (*Integer*), et deux autres objets de type ProbabilityDistribution pour représenter le bruit et le modèle de décision des plans.

Pour la reconnaissance de plan, seuls trois algorithmes sont nécessaires : un pour l'expansion de l'arbre de chaque particule, un pour l'initialisation des particules et un dernier pour l'actualisation des particules à chaque nouvelle observation.

Expansion de l'arbre d'une particule

Algorithme 1 : Expansion de l'arbre

```
1: function Expand(N = (a, E, r, f), EPL)
2:     if (s ϵ A) then
3:         f ←True
4:         return RNC(Bruit, a)
5:     end if
6:     e ← ProchaineAction(r, E)
7:     if (e ϵ E) then
8:         return Expand(E(e))
9:     end if
10:    if (e = ∅) then
11:        return ∅
12:    end if
13:    E(e) ← (e, ∅, RNC(DecisionModel, e), False)
14:    result ← Expand(E(e))
15:    if (For all s in r; C(s):f = True) then
16:        f = True
17:    end if
18:    return result
19: end function
```

Figure 5 : Algorithme d'expansion de l'arbre

Soit une particule $p = (O, P, PA)$ avec O un objectif, P un arbre de plan partiel et PA la prochaine action attendue.

Pour chaque nœud $N \epsilon P$, on définit $N = (a, E, r, f)$ avec a l'action représentée par ce nœud, E la liste d'enfants e ($e \in (A \cup NT)$) nécessaire pour accomplir cette action ($E = \emptyset$ si a est une action atomique), r la règle de production utilisée pour accomplir cette action et f un booléen qui représente si oui ou non l'action courante est finie (une action est finie si tous ses enfants E sont terminés ou si c'est une action atomique). Pour la racine (premier nœud), $a = O$.

Il est aussi nécessaire de définir deux fonctions :

- $int\ RNC(ProbabilityDistribution\ p, int\ i)$: renvoie un élément de p qui appartient à la liste d'éléments ayant pour clé i. Cet élément est choisi aléatoirement en prenant en compte les probabilités de p.

- $int\ ProchaineAction(Rule\ r, set < int > E)$: en fonction d'une règle de production *r* et des enfants *E*, renvoi une action suivante possible. Une action est possible si elle n'est pas finie et que toutes ses dépendances sont satisfaites. Si la fonction renvoie ∅ alors l'action courante est finie.

Initialisation

Algorithme 2 : Initialisation

```
1: function Init(Size)
2:     while (Size(Pop) < Size) do
3:         a ← Primitive(RNC(DecisionModel, Objectif))
4:         N ← (a, ∅, RNC(DecisionModel, a), False)
5:         PA ← Expand(N, EPL)
6:         p ← (a, N, PA)
7:         Pop[PA].insert(p)
8:     end while
9:     return Pop
10: end function
```

Figure 6 : Algorithme d'initialisation

Nous avons ici Pop qui est une population de particules représentée par une $map < int, set < Particule\ p >>$. Les particules sont regroupées en un set pour chaque prévision d'action. Ainsi, toutes les particules qui prédisent qu'une action *a* va se produire sont regroupées dans un set et mises dans la map avec pour clé, *a*.

Pour chaque particule, un objectif *a* est choisi aléatoirement grâce aux probabilités de distribution du modèle de décision. Puis grâce à ce même modèle de décision, nous choisissons un plan à suivre et nous créons l'arbre correspondant à ce plan où le nœud *N* est la racine. Puis, pour chaque particule, nous calculons l'estimation de la prochaine action. Nous avons donc maintenant tous les éléments pour créer la particule et l'insérer dans la population.

Filtrage

Algorithme 3 : Filtrage

```
1: function filtre(Pop, a ∈ A)
2:     size ← Size(Pop)
3:     FilteredPop ← Pop[a]
4:     Clear(Pop)
5:     while (Size(NewPop) < size) do
6:         p ← randomSelect(FilteredPop)
7:         p.PA = Expand(p.N, EPL)
8:         newPop[p.PA].insert(p)
9:     end while
10:    return newPop
11: end function
```

Figure 7 : Algorithme de filtrage

Ensuite, c'est très simple de filtrer les particules. On sauvegarde les particules correspondantes à l'observation, puis on supprime tout le reste. Il ne reste plus qu'à générer d'autres particules à partir de celles sauvegardées jusqu'à en avoir à nouveau le nombre voulu.

Résumé

Le module PARC que je viens de présenter était déjà présent lors de mon arrivée sur le projet. Mais étant nouveau, j'ai dû me familiariser avec le code qui est en pratique bien plus complexe que ces algorithmes. J'ai pu en profiter pour nettoyer et documenter entièrement cette partie du projet.

Cela a été pour moi un véritable défi de m'adapter en un temps restreint à un tout nouveau projet (surtout aussi complexe). Mais cela m'a permis de comprendre en détail tout le système du PARC et de programmer plus vite par la suite.

Cognitive Distress Management (CDM)

Le CDM est un module annexe à l'algorithme de reconnaissance de plan qui a pour but de détecter quand un comportement anormal se produit (quand l'utilisateur dévie de son plan ou quand il fait une erreur dans l'exécution de ce dernier).

C'est ce module qui permettra au robot Vigil de savoir quand agir. Car jusque-là, les algorithmes développés avaient pour seul but de capter et reconnaitre les activités de l'utilisateur. Ce module est conçu pour pouvoir fonctionner avec n'importe quel algorithme de reconnaissance de plan (les informations nécessaires à son exécution sont très génériques et n'importe quel algorithme de reconnaissance de plan est capable de produire ces données).

Lors de mon arrivée sur le projet, cet algorithme était déjà en partie programmé mais il n'était pas fini et il n'existait aucun résultat fiable prouvant son efficacité. Mon rôle a donc été de le reprogrammer proprement et de soumettre cet algorithme à une base de tests pouvant quantifier son efficacité.

Nous allons donc étudier dans un premier temps les mathématiques qui sont derrière le fonctionnement du CDM, puis voir comment cela a été implémenté. Dans un second temps, nous allons aborder la conception et l'implémentation de la base de tests, permettant la production de résultats complets et exploitables. Enfin, nous étudierons les résultats et les analyserons.

Il est d'abord nécessaire de définir ce qu'est comportement anormal et plus largement un comportement. Un comportement est un ensemble d'actions. Un comportement anormal est donc un comportement qui dévie du plan courant ou qui présente une anomalie dans son exécution. Pour détecter un comportement anormal, nous détectons donc les anomalies dans l'exécution des plans.

Cette détection était jusque-là un défi car les algorithmes de reconnaissance de plans étaient très peu rapides, et incapables de faire de la reconnaissance en temps réel. Pour remédier à cela, nous nous basons sur un algorithme de reconnaissance de plan par filtre à particule qui est bien assez rapide pour faire de la reconnaissance en temps réel. Il est donc possible de se pencher sur la détection d'anomalie.

Méthodes

Pour détecter un comportement anormal, nous détectons les anomalies dans l'exécution d'un plan. On utilise pour cela une approche probabiliste qui consiste à calculer $P(A_t)$, qui correspond à la probabilité d'avoir une anomalie à l'instant *t*. En appliquant la formule d'inférence de réseau bayésien dynamique :

$$P(A) = P(A|B)P(B) + P(A|\neg B)P(\neg B)$$
Avec $A = A_t$ et $B = A_{t-1}$

Nous obtenons donc : $P(A_t) = P(A_t|A_{t-1})P(A_{t-1}) + P(A_t|\neg A_{t-1})P(\neg A_{t-1})$

Nous émettons aussi deux hypothèses pour simplifier le calcul, mais qui découlent d'observation pratique et de bon sens. Premièrement, il n'est pas possible de détecter une anomalie alors qu'aucun plan n'est en cours ; donc $P(A_0) = 0$. Et nous considérons aussi que si un utilisateur dévie de son plan, alors il n'y reviendra pas. Cela se traduit par $P(A_t|A_{t-1}) = 1$. Avec ces deux hypothèses nous pouvons réduire la formule à ceci :

$$P(A_t) = P(A_{t-1}) + P(A_t|\neg A_{t-1})(1 - P(A_{t-1}))$$

Il ne reste donc que deux termes à calculer :

- $P(A_{t-1})$: c'est un terme défini par récurrence ; nous le connaissons déjà car nous l'avons calculé à l'itération précédente. C'est pour cela qu'il est important de définir que $P(A_0) = 0$.
- $P(A_t|\neg A_{t-1})$: c'est le terme clé que l'on va devoir calculer, et c'est ce sur quoi porte l'algorithme de détection de comportement anormaux. Il représente la probabilité d'avoir un comportement anormal sachant qu'il n'y en avait pas avant.

Pour calculer le terme $P(A_t|\neg A_{t-1})$, la méthode la plus simple serait de le définir come ci-dessous :

$$P(A_t|\neg A_{t-1}) = \sum_{e \in E} \sum_{a \in A} P(a \cap e \cap (a \notin e))$$

Qui se simplifie en : $P(A_t|\neg A_{t-1}) = \sum_{e \in E} \sum_{a \in A} (\delta_{a \notin e} P(a)P(e))$
Avec A l'ensemble des actions observées et E l'ensemble des plans possibles à l'instant *t*.

Cette formule est utilisable dans tous les algorithmes de reconnaissance de plan. Elle nécessite seulement la liste des actions observées et la liste des plans probables. Cependant elle ne fonctionne pas car dans la majorité des cas, $P(A_t|\neg A_{t-1}) > 0.5$.

Voici un exemple ; pour faire du thé, plusieurs ordres sont possibles :

Plans de e	Actions de e	P(e)
A	<Prendre une tasse, prendre un sachet thé, faire bouillir de l'eau, remplir la tasse>	0.2
B	< Prendre un sachet thé, prendre une tasse, faire bouillir de l'eau, remplir la tasse>	0.4
C	< Faire bouillir de l'eau, prendre une tasse, prendre un sachet thé, remplir la tasse>	0.4

Figure 8 : exemple d'anomalie

C'est vérifiable facilement avec cet exemple : soit trois plans A, B et C avec leur probabilité et les actions qui les composent :

Si la première action effectuée est l'action *prendre une tasse*, alors $P(A_t|\neg A_{t-1}) = 0.8$ car dans 80% des cas, *prendre une tasse* n'est pas le début du plan. Pour remédier à ce problème, il y a trois méthodes qui ont chacune leurs avantages et défauts et nous allons les voir maintenant.

Sum

La première méthode consiste seulement à imposer un seuil dans l'expression de $P(Ab_t)$ comme ceci :

$$P(A_t) = P(P(A_{t-1}) + P(A_t|\neg A_{t-1})\big(1 - P(A_{t-1})\big) > s)$$

Avec *s* un seuil généralement entre 95% et 99%.

Cela signifie qu'un comportement est considéré comme anormal si plus de *s*% des plans la considèrent comme anormal.

Min

Cette méthode se base sur de la logique floue et propose donc de remplacer la somme par le minimum comme ci-dessous :

$$\forall a \in A, \forall e \in E, \qquad P(A_t|\neg A_{t-1}) = min(\delta_{a \notin e} P(a)P(e))$$

Support

Cette méthode, en revanche, est spécifique à notre système de reconnaissance de plan (basé sur un filtre à particule) car il se base directement sur les particules.

Dans cet algorithme, nous utilisons *n* particules ayant chacune une prévision probable de la prochaine action basée sur le plan courant.

Nous utilisons ici aussi un seuil *s* pour définir à partir de quelle proportion de particules nous considérons qu'une action est considérée comme anormale.

$$P(A_t|\neg A_{t-1}) = P(\frac{nbPs}{bnP} < s)$$

Avec *nbPs* le nombre de particules qui supporte l'observation courante, et *nbP* le nombre total de particules.

Ici le seuil *s* varie souvent entre 1% et 5%.

Tests

Pour avoir une base de résultats correcte et exploitable, il est nécessaire de faire varier certains paramètres. Dans notre cas, nous avons fait varier la méthode mais aussi le seuil (appelé précision) quand il y en a un, ainsi que le bruit. Le bruit est le taux d'erreur de l'algorithme de reconnaissance de plan qui peut être dû à une action mal détectée ou à une mauvaise ligne de vue. Chaque test est réalisé dix fois sur cent bibliothèques de plans générées aléatoirement. Nous faisons aussi varier s'il y a un comportement anormal ou non, et la taille de la bibliothèque de plan.

Pour toutes les méthodes, nous faisons varier le bruit de 0% à 30%. Pour la méthode *Support*, nous avons fait varier la précision de 1% à 5% car c'est dans cet environ que l'algorithme est le plus efficace. En revanche, pour la méthode *Sum*, nous avons fait varier la précision de 1% à 15% car les variations entre 1% et 5% ne présentaient pas de grands changements. Il est donc intéressant d'observer les changements à une plus grande échelle. Les tests sont également faits sur quatre bibliothèques de plans différentes où l'on fait varier si une anomalie va se produire ou non, et le nombre d'actions atomiques différentes (variation entre 10 et 100).

Les bibliothèques de plans sont des bibliothèques génériques générées aléatoirement selon plusieurs paramètres : la prévision du bruit, le nombre d'objectifs, le nombre d'actions différentes, la taille et la profondeur des plans. À part le nombre d'actions qui varie pour augmenter ou diminuer l'impact du bruit, ces paramètres restent fixes.

Pour chaque méthode nous avons donc calculé l'impact du bruit sur l'efficacité, la proportion de faux négatif et de faux positif et la latence moyenne pour détecter une anomalie dans l'exécution du plan.

Implémentation

Pour l'implémentation du CDM, j'ai créé une classe abstraite 'CDM' et trois autres classes filles de CDM qui implémentent chaque méthode : CDMmin, CDMsum et CDMsupport.

Dans la classe CDM, on garde en mémoire l'état des particules dans une variable oldParticules pour que, si l'on détecte une anomalie, pouvoir reprendre la détection avant que cette dernière n'arrive. On stocke aussi dans un booléen si oui ou non une anomalie a été détecté. Deux autres variables proba et alpha sont présentes pour modéliser $P(A_t)$ et $P(A_t|\neg A_{t-1})$ respectivement. Des fonctions pour calculer $\forall a \in A, P(a)$ et $\forall e \in E, P(e)$ sont aussi présentes dans CDM car elles sont utiles pour les classes CDMmin et CDMsum. Une variable précision est présente dans les classes CDMsum et CDMsupport pour faire varier le seuil *s* nécessaire au calcul de $P(A_t|\neg A_{t-1})$.

Chaque classe fille de CDM présente une façon différente de calculer $P(A_t)$ et $P(A_t|\neg A_{t-1})$, elles ont donc chacune deux fonctions d'actualisation propre à ces termes.

Algorithme 4 : Actualisation du CDM

```
1: function update(CDM)
2:     updateAlpha(CDM.alpha)
3:     temp ← proba
4:     proba ← temp + alpha(1 – temp)
5:     updateAniormalBehavior(CDM.AB)
6:     if ( ! CDM.AB) then
7:         CDM.oldParticles ← s.particles
8:     end if
9:     return CDM
10: end function
```

Figure 9 : Algorithme d'actualisation du CDM

CDM contient aussi des références (pointeurs) à la bibliothèque de plan courante (EPL) et à la reconnaissance de plan (Solver *s*). Ici nous utilisons des pointeurs pour ne pas rajouter du temps de calcul à copier les éléments.

Résultats

Sum

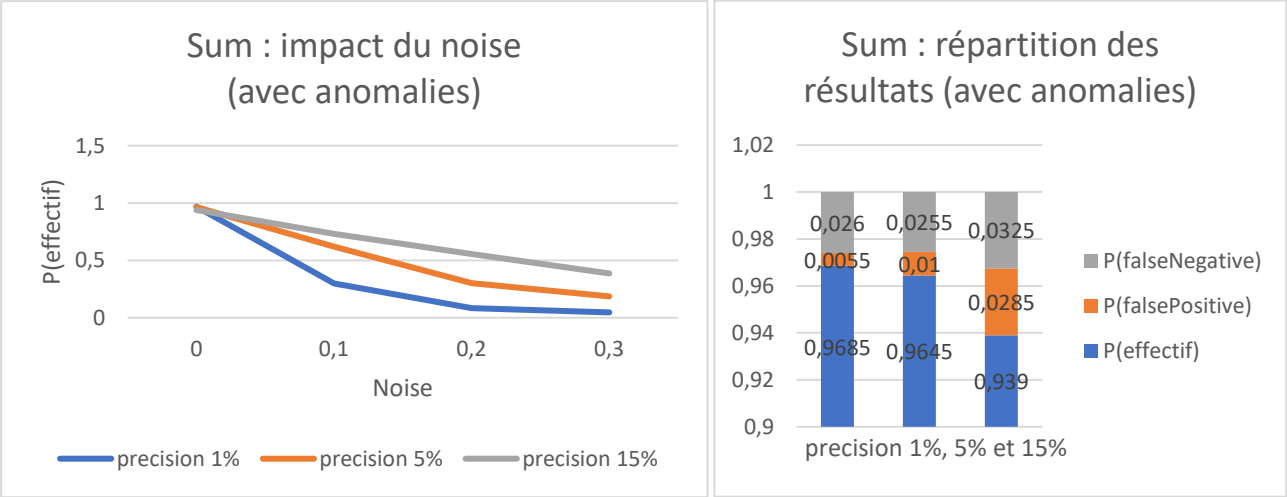


Figure 10 : résultat de la méthode Sum avec anomalies

Moyenne de latence : 0,795979545

Nous pouvons donc voir que cette méthode est très efficace (plus de 90%) mais qu'elle supporte mal le bruit : le taux de réussite chute fortement dès 10% de bruit. Mais ce problème peut être atténué par l'augmentation de la précision (en contrepartie d'un peu d'efficacité). La latence est correcte (inférieure à 1) ce qui permet d'alerter l'utilisateur rapidement après que l'anomalie se soit produite. Un autre problème de la méthode est qu'il y a plus de faux négatif que de faux positif. Nous préférons alerter l'utilisateur pour rien que de ne pas détecter quand il a réellement besoin d'aide.

Nous pouvons aussi voir le taux de faux positif quand il n'y a pas de comportements anormaux :

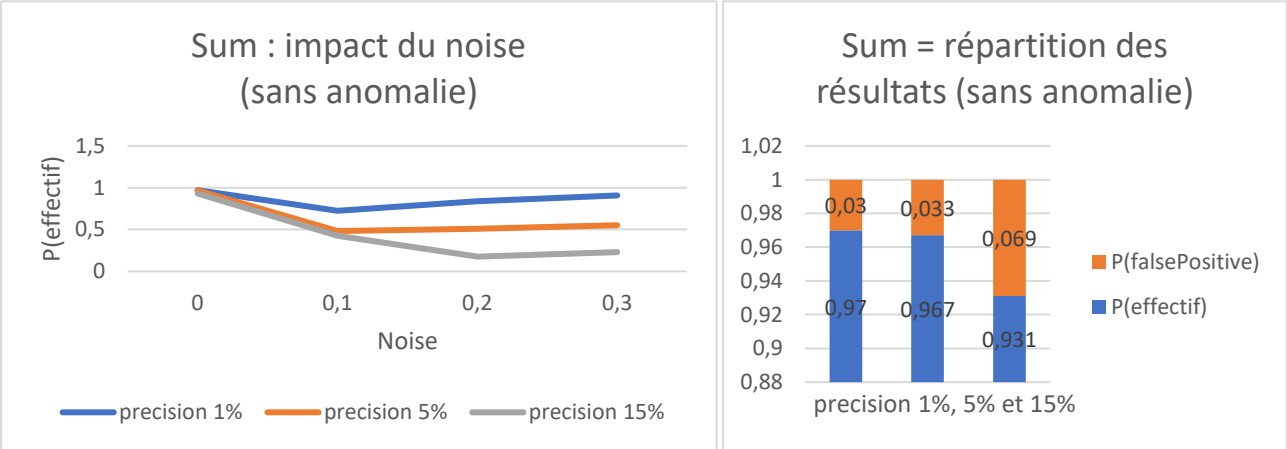


Figure 11 : résultat de la méthode Sum sans anomalie

Dans ce cas, le taux de faux positif reste très faible mais nous préférons resserrer la précision pour plus de résistance au bruit et plus d'efficacité.

Min

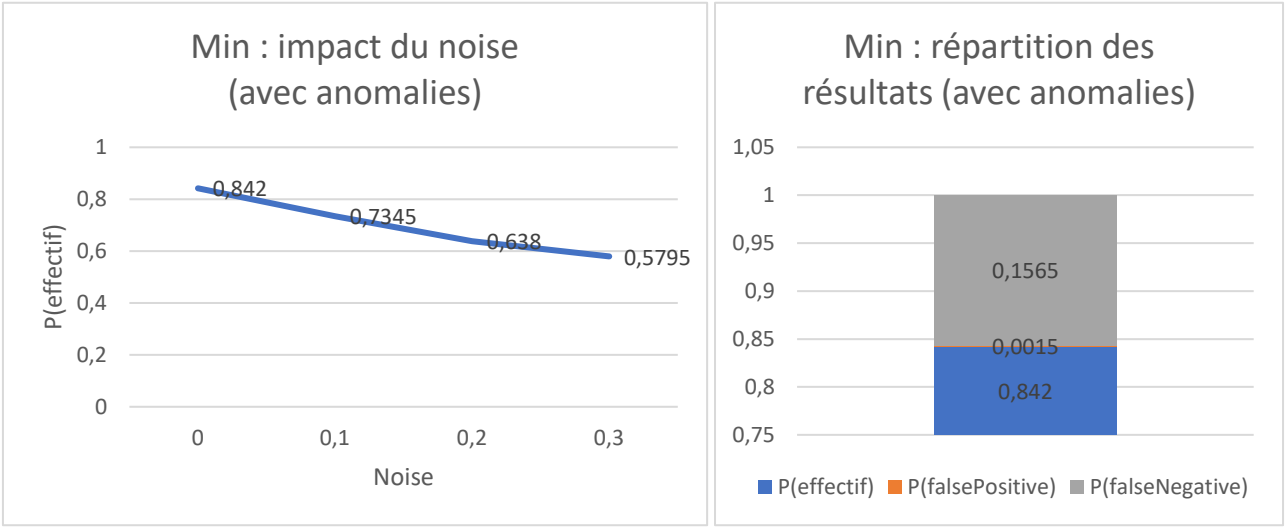


Figure 12 : résultat de la méthode Min avec anomalies

Moyenne de latence : 1,6639

La méthode min est bien moins efficace et les erreurs sont majoritairement des faux négatifs (à éviter absolument). De plus, la latence moyenne est supérieure à 1. Cela veut dire que pour aider l'utilisateur, nous devrons attendre une action après que ce dernier ait dévié de son plan. Mais l'avantage de cette méthode est sa résistance au bruit. L'efficacité reste supérieure à 50% même avec 30% de bruit. Cela en fait un très bon choix si la reconnaissance d'activité n'est pas assez efficace.

Voici maintenant les résultats sans anomalie dans l'exécution des plans :

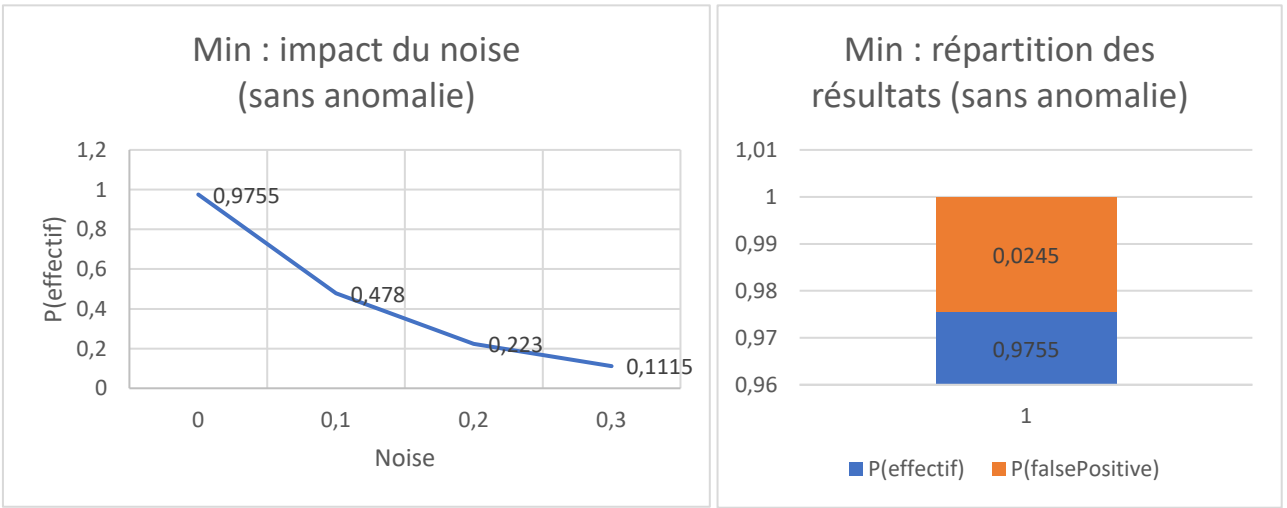
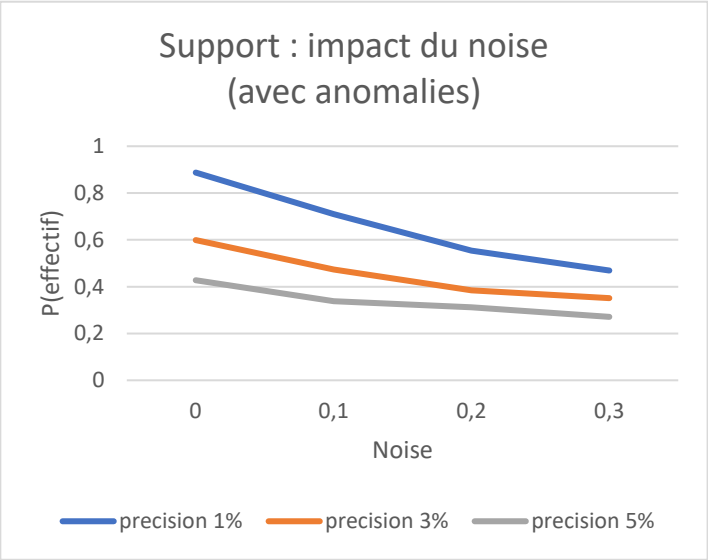


Figure 13 : résultat de la méthode Min sans anomali

Quand il n'y a pas d'anomalie, la méthode reste très efficace mais résiste beaucoup moins au bruit.

Support



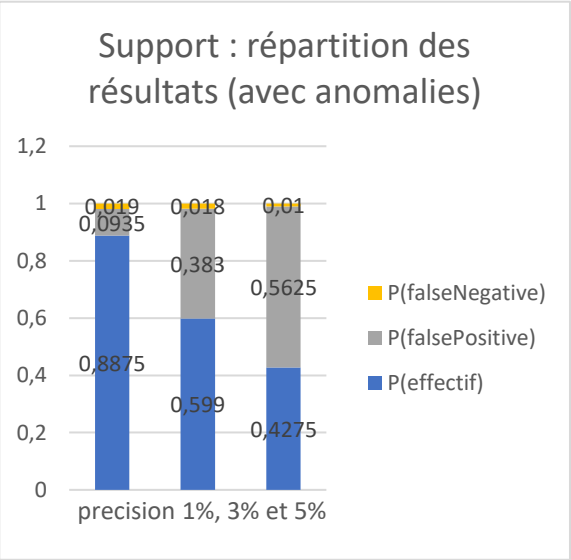
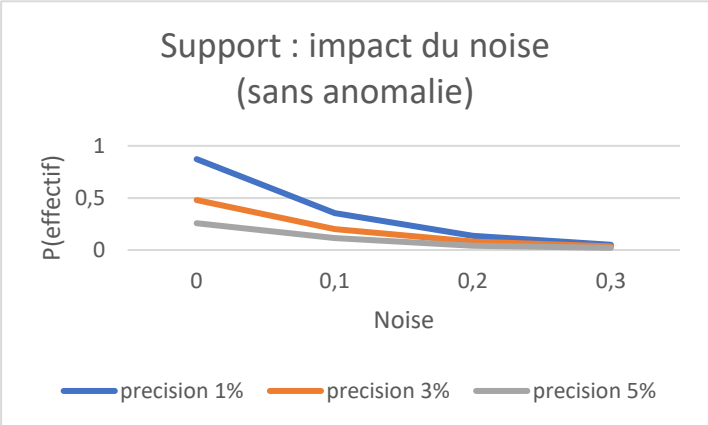


Figure 14 : résultat de la méthode Support avec anomalies

Moyenne de latence : 0,687503333

Cette méthode est la seule qui se base entièrement sur notre système de reconnaissance de plan par filtre à particule. Pourtant elle est moins efficace et peu résistante au bruit. De plus les autres précisions que 1% sont inutiles en vue des résultats. Cependant cette méthode est quand même très intéressante : Premièrement, elle a la latence la plus basse des trois méthodes. Si un comportement anormal est détecté, il le sera dans la plupart des cas dès qu'il a été effectué. Et deuxièmement, cette méthode a le plus bas taux de faux négatif. Si cette méthode se trompe c'est uniquement car elle a détecté à tort un comportement anormal.



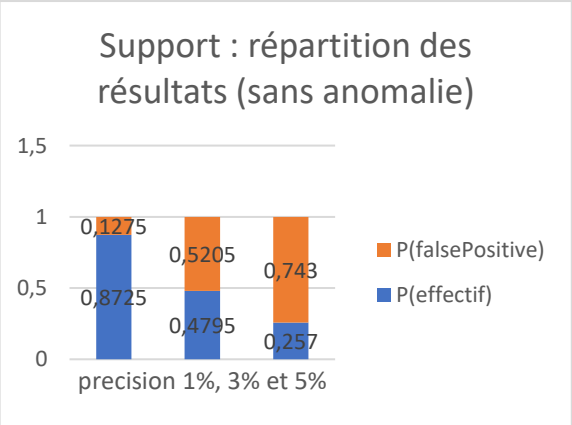


Figure 15 : résultat de la méthode Support sans anomalie

Cette caractéristique fait que si l'on arrive à augmenter son efficacité et à avoir un bruit proche de 0%, cette méthode est surement la plus viable dans des cas pratiques.

Cette méthode présente les mêmes défauts avec et sans anomalie : seule une précision de 1% est utilisable car les autres ne sont pas assez efficaces. Cependant, contrairement aux autres méthodes, varier la précision ne présente pas de dilemme. Il est donc possible de trouver un optimum où cette méthode sera la plus efficace.

Tous les résultats seront en Annexe sous forme de tableaux.

Résumé

Les résultats liés au test sont très concluants et toutes les méthodes sont efficaces en plus d'être très rapides (moins de 0.1 seconde de calculs). Chaque méthode a ses forces et ses faiblesses et est donc utilisable dans des contextes différents. Si l'environnement n'est pas favorable et que le bruit est élevé, nous allons préférer la méthode *Min*. Au contraire si notre but est d'avoir le plus d'efficacité sans bruit nous choisirons la méthode *Sum*. Enfin si nous voulons à tout prix éviter les faux négatif, la méthode *Support* sera parfaite.

Pour la suite, il faudra tester cet algorithme en pratique car les résultats actuels sont purement théoriques. Cela nécessite de l'implémenter au projet Vigil et de tester ses capacités dans des cas concrets de comportements anormaux.

Bilan

Pour le moment, mon stage de fin de DUT Informatique à Orsay a été très constructif. J'ai réussi à surmonter les difficultés, développer et tester un module complet du système PARC auquel j'avais été attribué. Ce stage m'a permis d'appliquer les compétences que j'ai accumulées tout au long de mon DUT telles que l'algorithmie ou le C++.

J'ai découvert le monde de la recherche et dû apprendre de nouvelles notions comme la bibliothèque QT par exemple. J'ai également beaucoup développé mes capacités de programmation en C++.

Travaillant depuis la France et avec un décalage horaire, j'ai dû travailler en complète autonomie en dehors des réunions hebdomadaires. Cela m'a permis de développer ma rigueur et ma capacité à m'adapter rapidement à un nouveau projet.

Grâce à ce stage, j'ai pu me faire une idée plus concrète du monde de la recherche et de son fonctionnement - que ce soit pour la partie recherche ou rédaction d'articles scientifiques. Même si les conditions n'étaient pas optimales, j'ai apprécié apporter ma contribution au projet Vigil.

Je me trouve dans un cas particulier où je n'ai pas encore terminé mon stage quand j'écris ce rapport. Je vais donc présenter mes objectifs futurs pour la suite.

Durant cette première partie de stage, j'ai pu programmer et tester un module de détection de comportement anormaux. Cependant, tous les résultats produits sont purement théoriques et aucun test pratique n'a été fait. Je souhaiterais implémenter le module CDM à l'application de Vigil. Ainsi je pourrai faire des tests pratiques sur l'efficacité du CDM. Cela me demandera de continuer d'apprendre la bibliothèque QT qui est la bibliothèque utilisée dans l'application.

Lexique

Int/Integer : nombrer entier.

Float : nombres réels.

String : chaine de caractères

Set : liste d'éléments.

Map : structure de données qui permet de stocker une liste d'éléments chacun identifié par un autre élément.

Classe : une classe représente une catégorie d'objets ayant des membres, méthodes et propriétés communes.

Réseau bayésien : un réseau bayésien est un modèle graphique probabiliste représentant un ensemble de variables aléatoires.

Réseau bayésien dynamique : un réseau bayésien dynamique permet de représenter l'évolution des variables aléatoires en fonction d'une séquence discrète, par exemple des pas temporels.

Table des illustrations

Figure 1 : Le pavillon Président-Kennedy (PK) de la faculté des sciences - UQAM

Figure 2 : représentation d'une bibliothèque de plan sous forme d'arbre

Figure 3 : Bibliothèque de plan

Figure 4 : représentation du filtre a particule

Figure 5 : Algorithme d'expansion de l'arbre

Figure 6 : Algorithme d'initialisation

Figure 7 : Algorithme de filtrage

Figure 8 : exemple d'anomalie

Figure 9 : Algorithme d'actualisation du CDM

Figure 10 : résultat de la méthode Sum avec anomalies

Figure 11 : résultat de la méthode Sum sans anomalie

Figure 12 : résultat de la méthode Min avec anomalies

Figure 13 : résultat de la méthode Min sans anomalie

Figure 14 : résultat de la méthode Support avec anomalies

Figure 15 : résultat de la méthode Support sans anomalie

Annexes

AB : False

nbActions : 10

methode : min

noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
0	0,955	0,045	0	0,987	0
0,1	0,504	0,496	0	0,883	0
0,2	0,253	0,747	0	0,788	0
0,3	0,159	0,841	0	0,739	0

methode : sum

precision : 1

noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
0	0,947	0,053	0	0,984	0
0,1	1	0	0	0,871	0
0,2	1	0	0	0,818	0
0,3	1	0	0	0,713	0

precision : 3

noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
0	0,95	0,05	0	0,982	0
0,1	0,879	0,121	0	0,894	0
0,2	0,989	0,011	0	0,783	0
0,3	1	0	0	0,723	0

precision : 5

noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
0	0,942	0,058	0	0.978	0
0,1	0,6	0,4	0	0,892	0
0,2	0,885	0,115	0	0,783	0
0,3	0,99	0,01	0	0,73	0

precision : 10

noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
0	0,928	0,072	0	0,971	0
0,1	0,429	0,571	0	0,875	0
0,2	0,448	0,552	0	0,8	0
0,3	0,847	0,153	0	0,731	0

precision : 15

noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
0	0,904	0,096	0	0,955	0
0,1	0,44	0,56	0	0,869	0
0,2	0,227	0,773	0	0,769	0
0,3	0,421	0,579	0	0,735	0

precision : 20

noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
0	0,865	0,135	0	0,928	0
0,1	0,379	0,621	0	0,842	0
0,2	0,188	0,812	0	0,774	0
0,3	0,205	0,795	0	0,722	0

methode : support

precision : 1

noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
0	0,899	0,101	0	0,961	0
0,1	0,379	0,621	0	0,851	0
0,2	0,166	0,834	0	0,763	0
0,3	0,068	0,932	0	0,678	0

precision : 3

noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
0	0,698	0,302	0	0,881	0
0,1	0,311	0,689	0	0,769	0
0,2	0,116	0,884	0	0,703	0
0,3	0,06	0,94	0	0,66	0

precision : 5

noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
0	0,451	0,549	0	0,777	0
0,1	0,201	0,799	0	0,696	0
0,2	0,068	0,932	0	0,627	0
0,3	0,037	0,963	0	0,578	0

AB :	False
nbActions :	100

methode :	min					
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,996	0,004	0	0,998	0
	0,1	0,452	0,548	0	0,908	0
	0,2	0,193	0,807	0	0,825	0
	0,3	0,064	0,936	0	0,763	0

methode :	sum					
	precision :	1				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,993	0,007	0	1	0
	0,1	0,448	0,552	0	0,917	0
	0,2	0,674	0,326	0	0,833	0
	0,3	0,813	0,187	0	0,712	0
	precision :	3				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,995	0,005	0	1	0
	0,1	0,383	0,617	0	0,902	0
	0,2	0,2	0,8	0	0,836	0
	0,3	0,282	0,718	0	0,757	0
	precision :	5				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,992	0,008	0	0,999	0
	0,1	0,364	0,636	0	0,884	0
	0,2	0,131	0,869	0	0,836	0
	0,3	0,11	0,89	0	0,752	0
	precision :	10				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,988	0,012	0	0,996	0
	0,1	0,39	0,61	0	0,895	0
	0,2	0,124	0,876	0	0,803	0
	0,3	0,031	0,969	0	0,725	0
	precision :	15				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,958	0,042	0	0,968	0
	0,1	0,413	0,587	0	0,897	0
	0,2	0,125	0,875	0	0,804	0
	0,3	0,039	0,961	0	0,732	0
	precision :	20				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,739	0,261	0	0,75	0
	0,1	0,282	0,718	0	0,631	0
	0,2	0,09	0,91	0	0,495	0
	0,3	0,023	0,977	0	0,47	0

methode :	support					
	precision :	1				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,846	0,154	0	0,883	0
	0,1	0,329	0,671	0	0,8	0
	0,2	0,107	0,893	0	0,739	0
	0,3	0,032	0,968	0	0,657	0
	precision :	3				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,261	0,739	0	0,423	0
	0,1	0,093	0,907	0	0,412	0
	0,2	0,045	0,955	0	0,388	0
	0,3	0,009	0,991	0	0,345	0
	precision :	5				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,063	0,937	0	0,257	0
	0,1	0,028	0,972	0	0,266	0
	0,2	0,011	0,989	0	0,253	0
	0,3	0,005	0,995	0	0,267	0

AB :	True					
nbActions :	10					
methode :	min					
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,805	0,002	0,193	0,835	2,61
	0,1	0,766	0,118	0,116	0,774	2,457
	0,2	0,721	0,183	0,096	0,737	2,426
	0,3	0,684	0,264	0,052	0,669	2,481
methode :	sum					
	precision :	1				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,938	0,011	0,051	0,848	1,489
	0,1	0	0	1	0,794	0
	0,2	0	0	1	0,726	0
	0,3	0	0	1	0,66	0
	precision :	3				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,937	0,012	0,051	0,841	1,443
	0,1	0,126	0,027	0,847	0,777	1,476
	0,2	0,004	0,003	0,993	0,732	0,5
	0,3	0	0	1	0,664	0
	precision :	5				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,935	0,016	0,049	0,828	1,517
	0,1	0,553	0,1	0,347	0,766	1,72
	0,2	0,087	0,02	0,893	0,738	1,425
	0,3	0,002	0,003	0,995	0,66	2
	precision :	10				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,932	0,024	0,044	0,852	1,339
	0,1	0,783	0,177	0,04	0,788	1,682
	0,2	0,439	0,173	0,388	0,736	1,818
	0,3	0,068	0,044	0,888	0,682	1,382
	precision :	15				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,905	0,032	0,063	0,822	1,309
	0,1	0,785	0,185	0,03	0,74	1,703
	0,2	0,633	0,277	0,09	0,722	1,698
	0,3	0,389	0,185	0,426	0,676	1,856
	precision :	20				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,904	0,054	0,042	0,787	1,408
	0,1	0,752	0,21	0,038	0,741	1,495
	0,2	0,686	0,288	0,026	0,718	1,78
	0,3	0,54	0,293	0,167	0,639	1,969
methode :	support					
	precision :	1				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,929	0,037	0,034	0,826	1,41
	0,1	0,802	0,175	0,023	0,76	1,546
	0,2	0,657	0,323	0,02	0,694	1,715
	0,3	0,612	0,369	0,019	0,643	1,794
	precision :	3				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,819	0,145	0,036	0,748	1,056
	0,1	0,687	0,302	0,011	0,676	1,223
	0,2	0,55	0,441	0,009	0,669	1,38
	0,3	0,517	0,475	0,008	0,59	1,358
	precision :	5				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,67	0,31	0,02	0,627	0,9478
	0,1	0,513	0,474	0,013	0,605	1,027
	0,2	0,484	0,514	0,002	0,567	0,9463
	0,3	0,403	0,594	0,003	0,538	1,127

AB :	True					
nbActions :	100					
methode :	min					
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,879	0,001	0,12	0,902	1,139
	0,1	0,703	0,235	0,062	0,824	1,061
	0,2	0,555	0,423	0,022	0,76	1,009
	0,3	0,475	0,516	0,009	0,711	0,9453
methode :	sum					
	precision :	1				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,999	0	0,001	0,897	0,1291
	0,1	0,598	0,24	0,162	0,849	0,1605
	0,2	0,166	0,139	0,695	0,75	0,1687
	0,3	0,093	0,079	0,828	0,693	0,1505
	precision :	3				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,992	0,002	0,006	0,903	0,1159
	0,1	0,692	0,307	0,001	0,83	0,1156
	0,2	0,484	0,408	0,108	0,734	0,2128
	0,3	0,256	0,441	0,303	0,692	0,2617
	precision :	5				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,994	0,004	0,002	0,883	0,1157
	0,1	0,688	0,311	0,001	0,812	0,1424
	0,2	0,515	0,485	0	0,748	0,1786
	0,3	0,37	0,557	0,073	0,681	0,2351
	precision :	10				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,992	0,006	0,002	0,877	0,1381
	0,1	0,696	0,304	0	0,841	0,1509
	0,2	0,482	0,518	0	0,736	0,2116
	0,3	0,4	0,6	0	0,683	0,21
	precision :	15				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,973	0,025	0,002	0,869	0,1223
	0,1	0,679	0,321	0	0,799	0,1487
	0,2	0,48	0,52	0	0,726	0,1979
	0,3	0,383	0,617	0	0,666	0,2637
	precision :	20				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,763	0,236	0,001	0,657	0,1048
	0,1	0,47	0,53	0	0,538	0,1383
	0,2	0,341	0,659	0	0,482	0,1496
	0,3	0,261	0,739	0	0,427	0,1916
methode :	support					
	precision :	1				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,846	0,15	0,004	0,765	0,1111
	0,1	0,619	0,381	0	0,719	0,1179
	0,2	0,452	0,548	0	0,649	0,1704
	0,3	0,326	0,674	0	0,624	0,1933
	precision :	3				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,379	0,621	0	0,424	0,06069
	0,1	0,26	0,74	0	0,385	0,05769
	0,2	0,218	0,782	0	0,393	0,06881
	0,3	0,185	0,815	0	0,322	0,07027
	precision :	5				
	noise	P(effective)	P(falsePositive)	P(falseNegative)	P(GoalFound)	Moy(latency)
	0	0,185	0,815	0	0,243	0,03784
	0,1	0,162	0,838	0	0,241	0,02469
	0,2	0,14	0,86	0	0,249	0,03571
	0,3	0,139	0,861	0	0,234	0,02158