Amélioration continue d'une chaîne de traitement de documents avec l'apprentissage par renforcement

Esther Nicart[1,2], Bruno Zanuttini[2], Bruno Grilhères[1], Patrick Giroux[1]

[1] IPCC, Airbus Defence and Space, France, prenom.nom@airbus.com

[2] MAD, GREYC, Université de Caen, Basse-Normandie, France, prenom.nom@unicaen.fr

Résumé : Nous nous intéressons au problème de l'amélioration continue d'une chaîne de traitement de documents, visant à extraire des événements dans des documents provenant de sources ouvertes. Il s'agit de tirer parti des corrections effectuées par les opérateurs humains pour que la chaîne de traitement apprenne de ses erreurs, et s'améliore de façon générale.

Nous appliquons l'apprentissage par renforcement (en l'occurrence, le *Q-Learning*) à ce problème, où les actions sont les services d'une chaîne de traitement d'extraction de l'information. L'objectif est de profiter du *feedback* utilisateur pour permettre au système d'apprendre la configuration idéale des services (ordonnancement, *gazetteers* et règles d'extraction) en fonction des caractéristiques des documents à traiter (langue, type, etc.). Nous menons de premières expériences avec des données de *feedback* générées automatiquement à partir d'un oracle, et les résultats sont encourageants.

Mots-clés : Intelligence artificielle, Apprentissage par renforcement, Extraction et gestion des connaissances, Interaction homme-machine, Renseignement d'origine sources ouvertes (ROSO)

1 Introduction

Nous nous intéressons au problème générique de l'amélioration continue d'une chaîne de traitement de documents, plus précisément d'extraction d'événements d'intérêt. L'application qui nous intéresse particulièrement est le renseignement à partir de sources ouvertes. Dans cette application, des documents provenant essentiellement du *web* sont fournis en continu à une chaîne de traitement, qui vise à extraire des événements (par exemple, une attaque terroriste) et leurs caractéristiques (date, lieu, acteurs, etc.) et à les intégrer à une base de données.

Dans de telles applications, il est clair que l'extraction ne peut pas être parfaite. On peut ainsi imaginer qu'une dépêche relatant « le *bombardement*, par des ions, d'une *cible* d'or par la physique *atomique* lors d'une *manifestation* pour la fête de la science », puisse induire une chaîne de traitement en erreur et lui fasse insérer dans la base un attentat à l'arme atomique. Par ailleurs, la volonté de traiter des documents provenant du monde entier entraîne le besoin de traiter des documents dans des langues très diverses, pour lesquelles des dictionnaires peuvent être de qualité très variable. Pour toutes ces raisons, le cadre typique implique des opérateurs humains, qui peuvent corriger *a posteriori* les événements placés automatiquement en base de données.

L'application qui nous intéresse utilise la plateforme WebLab (2015) pour le renseignement d'origine sources ouvertes. La chaîne de traitement, définie par des experts, consiste en un enchaînement figé (mais potentiellement conditionnel) de traitements atomiques, tels que l'extraction de la langue ou du format, la traduction, la détection d'événements en utilisant des mots ou verbes déclencheurs, etc. Aucun mécanisme ne permet d'améliorer la chaîne au fil du temps.

Notre objectif est de combler ce manque, en fournissant un mécanisme d'amélioration continue de la chaîne de traitement, tirant parti des retours (du *feedback*) exprimés implicitement par les opérateurs lorsqu'ils corrigent des événements dans la base de données. Il s'agit de faire en sorte que la chaîne « apprenne de ses erreurs ». Ainsi, le système pourrait apprendre des règles telles que « si le document est en breton, il est préférable de le traduire d'abord en français puis d'extraire des événements, plutôt que d'appliquer directement une phase d'extraction sur le breton », en constatant que des corrections sont souvent apportées sur les événements extraits par la deuxième option.

À moyen terme, nous visons la prise en compte directe du *feedback* de l'utilisateur. Son expertise (le *feedback*) est manifestée par les traces de ses actions (Bratko & Suc, 2003). Ces traces seront captées à travers l'interface graphique qui donne les détails des événements en synthèse. La chaîne de traitement restera alors une « boîte noire » pour l'utilisateur. Il s'agira de retours qualitatifs, en particulier sur les événements extraits (« corrigé », « consulté et non corrigé », « non consulté », etc.).

Nous restreignons ici le cadre, en supposant que le système reçoit un *feedback* quantitatif sur la qualité des extractions. Nous proposons une formalisation du problème en apprentissage par renforcement, et rendons compte de premières expériences très encourageantes. Pour celles-ci, le *feedback* est basé sur une distance entre les événements désiré et extrait et donc, implicitement, sur le temps qui serait nécessaire à la correction des erreurs.

Culotta *et al.* (2006) ont montré l'intérêt de solliciter des corrections à l'utilisateur, et de guider la mise à jour du modèle. Pourtant le modèle pourrait être bon, mais mal appliqué, et les utilisateurs n'ont pas l'expertise pour modifier le système. Chai *et al.* (2009) ont essayé de combler ce manque d'expertise en proposant un langage permettant aux utilisateurs de corriger directement l'application d'extraction de l'information sans l'intervention d'un expert technique, mais cela suppose que l'amélioration s'applique à tous les documents à traiter, c'est-à-dire, dans une chaîne de traitement figée. Or, les documents de renseignement sont hétérogènes, et sont traités dans des grands volumes.

Nous proposons de permettre à l'utilisateur de se distancer complètement du modèle, en offrant un système modulaire [*cf.* Fromherz *et al.* (2003) qui construisent de chaînes de façon adaptative et modulaire pour des photocopieurs Xerox], qui apprend à modifier son propre comportement en temps réel [*cf.* Doucy *et al.* (2008) qui modifient également une chaîne de traitement à la volée], à partir du *feedback* offert par l'utilisateur [*cf.* Dupont *et al.* (2011) qui montrent l'utilisation de RL pour la sélection d'outils de recherche basée sur l'analyse des actions de l'utilisateur], et qui plus est, s'adapte à chaque document source unique.

2 La plateforme WebLab

La chaîne de traitement qui motive notre travail s'appuie sur la plateforme *open-source* WebLab (2015). WebLab intègre des services *web* qui peuvent être interchangés ou permutés afin de créer une chaîne de traitement. Cette chaîne peut ensuite être utilisée pour analyser des documents multimédia *open-source*, et en extraire l'information.

Une chaîne de traitement typique de WebLab (Figure 1) commence par convertir le document source en une ressource XML. Cette ressource est ensuite transmise de service en service. Chaque service analyse le contenu de la ressource telle qu'il la reçoit, et l'enrichit avec des annotations. Enfin, les résultats sont stockés pour consultation par l'utilisateur.

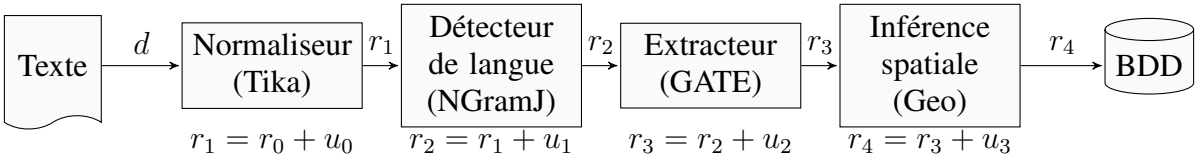


FIGURE 1 – Une chaîne WebLab typique : un document, d, est converti en une ressource XML, r_0, et des annotations, u_j, sont ajoutées par chaque service. Enfin, les résultats sont stockés.

```
1  <resource type="Document" uri="weblab:aaa">
2    <annotation uri="weblab:aaa#a0">
3      <Description about="weblab:aaa">
4        <wp:hasNativeContent resource="file:weblab.content"/>
5        <wp:hasOriginalFileSize>255</wp:hasOriginalFileSize>
6        <dc:source>documents/event.txt</dc:source>
7        <wp:hasOriginalFileName>event.txt</wp:hasOriginalFileName>
8        <dcterms:modified>2015-02-14T19:52:21+0100</dcterms:modified>
9        <wp:hasGatheringDate>2015-02-10T00:11:00+0200</wp:hasGatheringDate>
10     </Description>
11   </annotation>
12   <annotation uri="weblab:aaa#a1">
13     <Description about="weblab:aaa">
14       <wp:isProducedBy resource="weblab:tika"/>
15       <dc:format>text/plain</dc:format>
16     </Description>
17   </annotation>
18   <annotation uri="weblab:aaa#a2">
19     <Description about="weblab:aaa">
20       <wp:isProducedBy resource="weblab:ngramj"/>
21       <dc:language>en</dc:language>
22     </Description>
23   </annotation>
24   <mediaUnit type="wl:Text" uri="weblab:aaa#0">
25     <content>4/29/1971: In a series of two incidents that might have been part of a multiple attack, suspected members of the
         Chicano Liberation Front bombed a Bank of America branch in Los Angeles, California, US. There were no casualties but
       the building sustained $1,600 in damages.</content>
26   </mediaUnit>
27 </resource>
```

FIGURE 2 – Flux XML simplifié d'une ressource WebLab, qui montre les informations extraites telles que le format (ligne 15), la langue (ligne 21) et le contenu original (ligne 25).

Exemple 1
Considérons le document textuel suivant :

> `4/29/1971: In a series of two incidents that might have been part of a multiple attack, suspected members of the Chicano Liberation Front bombed a Bank of America branch in Los Angeles, California, US. There were no casualties but the building sustained $1,600 in damages.`

La ressource XML (simplifiée) de la Figure 2 est produite après le passage du document par le normaliser, *Tika (2015), qui a ajouté l'annotation « text/plain » (ligne 15) et le contenu original, et le détecteur de la langue, NGramJ (2015), qui a ajouté la langue « en » (ligne 21).*

Nous considérons l'utilisation d'une telle chaîne pour l'extraction d'événements d'intérêt pour la veille économique, stratégique, ou militaire. En travaillant avec WebLab, nous nous situons dans la continuité du travail de Serrano (2014), qui propose la définition suivante d'un événement, formalisé dans l'ontologie *WOOKIE*. Notre travail est indépendant de WebLab et *WOOKIE*, et nous aurions pu utiliser une autre définition, par exemple, celle de van Hage *et al.* (2011).

Définition 1 (Un événement)
Un événement E *est un quadruplet* $E =< C, T, S, A >$*, où :*

- $C \subseteq \mathbb{C}$ *est la dimension conceptuelle de* E*, donnée par un ensemble d'atomes pris dans un domaine* $\mathbb{C}$ *commun à tous les événements ;*
- T *est la dimension temporelle de* E*, c'est-à-dire la date à laquelle* E *est survenu (potentiellement ambiguë, telle que « mardi dernier ») ; pour modéliser l'ambiguïté, on prend* $T \subseteq \mathbb{T}$*, où* $\mathbb{T}$ *est l'ensemble des dates ;*
- S *est la dimension spatiale de* E*, potentiellement ambiguë également, avec* $S \subseteq \mathbb{S}$ *;*
- A *est la dimension agentive de* E*, càd l'ensemble des participants impliqués (*$A \subseteq \mathbb{A}$*).*

Si la définition est générale, on s'intéresse dans le cadre de cet article à des domaines précis : $\mathbb{C}$ est un ensemble fixé et fini d'atomes dans *WOOKIE* ; $\mathbb{T}$ est l'ensemble de toutes les « dates » qui peuvent être extraites, par exemple « mardi dernier », « 2001 », « 2001/9/11, 8:46 » ; $\mathbb{S}$ est l'ensemble des entités utilisées par GeoNames (2015) ; enfin, $\mathbb{A}$ est l'ensemble (infini) de tous les participants pouvant être extraits, vus comme des chaînes de caractères.

Exemple 2 (suite de l'exemple 1)
Le document au dessus donnera lieu à l'extraction d'un événement $E =< C, T, S, A >$ *avec* $C = \{$*AttackEvent*, *BombingEvent*$\}$, $T = \{$*4/29/1971*$\}$, $S = \{$*Los Angeles*, *California*, *United States*, *America*$\}$, $A = \{$*Chicano Liberation Front*, *Bank of America*$\}$.

3 Apprentissage par renforcement

Pour atteindre notre objectif, nous appliquons les techniques de l'*apprentissage par renforcement* (*Reinforcement Learning*, RL). Pour une introduction détaillée au sujet, nous renvoyons le lecteur à Sutton & Barto (1998), mais nous en rappelons les grands principes dans cette section.

En RL, l'apprenant reçoit une récompense, basée sur les résultats des actions qu'il a choisies. Plus les résultats sont proches des objectifs, plus la récompense est élevée. Le système essaie de maximiser ces récompenses, typiquement en *exploitant* ce qu'il connaît déjà pour continuer à recevoir de bonnes récompenses, et en *explorant* de nouvelles actions avec l'espoir d'obtenir des récompenses encore plus importantes. Prenons l'exemple d'un robot qui doit naviguer sur une grille. Son objectif est d'atteindre une case spécifique, qui est la seule à donner une récompense. Typiquement, au fil des épisodes, il renforcera la valeur des cases depuis lesquelles il sera arrivé rapidement au but, et apprendra ainsi le chemin idéal.

Le RL est généralement formalisé comme un processus de décision markovien (*Markov Decision Process*, MDP). Un tel processus modélise l'environnement en termes d'*états*, dans lesquels des *actions* sont possibles, qui mènent à d'autres états de manière stochastique. Le fait que l'environnement soit dans un état donné à un certain instant apporte une récompense immédiate à l'agent. L'objectif d'un apprenant est de choisir ses actions de façon à maximiser son espérance de récompenses cumulées, sans connaître, initialement, ni les distributions sur les états résultant d'une action, ni les récompenses associées aux états. Bien entendu, ce cadre générique admet de nombreuses variantes (pour un aperçu récent, voir Szepesvári (2010)).

Définition 2 (MDP)
Un processus de décision markovien *(Puterman, 1994) est un 5-uplet (*S, A, P, R, γ*), avec*

- S *un ensemble (fini, discret) d'états possibles de l'environnement,*

- A *un ensemble (fini, discret) d'actions (que l'agent peut effectuer),*
- P *un ensemble de distributions* $\{P_a(s,\cdot) \mid s \in S, a \in A\}$ *;* $P_a(s,s')$ *est la probabilité que l'environnement soit dans l'état* s' *après que l'agent a effectué l'action* a *en* s*,*
- R *une fonction de récompense, que nous supposons définie sur les états ;* $R(s)$ *est la récompense obtenue par l'agent pour se trouver dans l'état* s*,*
- $\gamma \in [0,1]$ *un facteur d'atténuation, qui contrôle l'importance des récompenses espérées dans le futur, relativement aux récompenses espérées dans l'immédiat.*

Dans le cadre du RL, l'agent (apprenant) connaît initialement seulement les espaces d'états et d'actions S, A, ainsi que le facteur γ. À tout instant t, il connaît l'état courant s_t de l'environnement, et choisit une action a_t. L'environnement passe dans un état s_{t+1} tiré selon la distribution $P_{a_t}(s_t,\cdot)$, et l'agent est informé de l'état s_{t+1} et de la récompense $r_{t+1} = R(s_{t+1})$. Le processus continue en s_{t+1}. L'agent doit, au fil de ces interactions avec l'environnement, apprendre une série de *politiques* $\pi_0, \pi_1, \ldots, \pi_t, \ldots$, une *politique* $\pi_t : S \to A$ donnant, pour l'instant t, l'action $\pi_t(s)$ à effectuer si l'état courant est $s_t = s$, pour tout $s \in S$. Son objectif est à tout instant de maximiser l'espérance de la récompense cumulée, c'est-à-dire l'espérance de la quantité $\sum_{t'=t}^{\infty} \gamma^t R(s_t)$.

Dans l'exemple du robot, à l'instant t, l'état courant s_t est la case sur laquelle il se trouve, et son action a_t est prise parmi *nord*, *ouest*, etc. La probabilité $P_{a_t}(s_t, s')$ qu'il arrive sur la case s' à l'instant suivant dépend de sa case de départ s_t et de l'action a_t.

Il n'est pas si trivial de définir des états et actions dans une chaîne de traitement, pourtant, il semble naturel d'utiliser le RL sur notre problématique. Les seules informations connues avant de commencer une chaîne de traitement sont les services disponibles, leurs paramètres, et les états potentiels des documents et du système (*cf.* section 2). L'apprenant ne connaît ni la forme, ni le contenu des documents à l'avance, ni si une extraction sera possible, donc ses décisions sont prises dans l'incertain. Nous voulons que le système apprenne une série de *politiques* adaptées aux besoins de l'utilisateur, pour améliorer en continu l'extraction des événements.

De nombreux algorithmes ont été proposés dans la littérature pour les problèmes de RL. Dans cet article, nous utilisons une des approches les plus standards, le *Q-learning* (Watkins, 1989), avec une exploration ϵ-gloutonne. Cette approche consiste à maintenir, pour chaque couple état/action (s,a), une valeur notée $\hat{Q}(s,a)$ qui représente intuitivement l'estimation courante, par l'agent, de l'espérance de récompense s'il se trouve dans s, exécute a, puis suit une *politique* optimale. En résumé, lorsque l'agent est dans l'état s_t, choisit a_t, se retrouve en s_{t+1} et reçoit une récompense r_{t+1}, il met à jour son estimation de la valeur $\hat{Q}(s_t,a_t)$ de la manière suivante : $\hat{Q}(s_t,a_t) \leftarrow (1-\alpha)\hat{Q}(s_t,a_t) + \alpha\gamma(r_{t+1} + \gamma\, max_{a_{t+1}} Q(s_{t+1},a_{t+1}))$ où α, le *taux d'apprentissage*, est un coefficient dans $[0,1]$ qui fixe l'importance de la dernière expérience $(r_{t+1} + \gamma\, max(\ldots))$ par rapport à l'expérience déjà accumulée (l'ancienne valeur de $\hat{Q}(s_t,a_t)$).

Enfin, dans le Q-learning avec exploration ϵ-gloutonne, le *taux d'exploration* $\epsilon \in [0,1]$ règle le dilemme exploitation/exploration de la manière suivante. À chaque pas de temps t, l'agent tire un nombre aléatoire dans $[0,1]$; s'il fait entre 0 et ϵ, alors il choisit une action aléatoirement (il *explore*) ; sinon, l'agent *exploite* et choisit simplement l'action a qui maximise $\hat{Q}(s_t,a)$.

Le *Q-learning* est un algorithme simple, dont les paramètres α et ϵ peuvent être réglés de façon intuitive, et c'est pourquoi nous l'utilisons dans la suite. Toutefois, notre contribution consiste à modéliser le problème de l'amélioration continue d'une chaîne de traitement comme un problème de RL, et tout algorithme pour ce problème pourrait également être utilisé.

4 Amélioration continue via l'apprentissage par renforcement

Dans la pratique, la chaîne de traitement utilisée est complexe. Elle est écrite et calibrée par des experts qui choisissent les services constituant la chaîne, leur ordonnancement et leurs paramètres (par exemple, les *gazetteers* de mots déclencheurs pour les services de détection d'événements). L'ordonnancement peut être conditionnel (par exemple, si le document est en format pdf, passer au service 1 pour le convertir en XML, et au service 2 sinon), mais il est figé.

Malgré l'expertise, il est très difficile d'obtenir une chaîne parfaite, parce que l'utilisation des documents *open source* provenant du *web* apporte des difficultés : leurs format et contenu ne sont pas standards, les pages sources elles-mêmes ne sont pas contrôlables, les urls changent, ou sont piratés, et il y a du « bruit » (publicité par exemple). On observe des erreurs d'extraction pouvant être des événements d'intérêt manqués, des événements mal extraits (les informations non connexes dans la même phrase associées faussement, par exemple).

Par exemple, l'utilisateur voudrait de l'information sur les accords entre pays. Il est impossible de dire avec certitude que le mot « alliance » dans un document y fait référence. Ce n'est qu'après l'extraction de l'événement déclenchée par le mot « alliance » que l'utilisateur se rend compte que la page parle de mariages, par exemple. Même si le document provient d'un journal politique, il se peut que l'on parle d'une coalition entre partis politiques, ou que les filtres de publicité n'aient pas réussi à attraper une vente de bagues. Le mot « union » a pu être utilisé au lieu d'« alliance », et l'événement n'a pas été reconnu. Avec ces incertitudes, les experts qui paramètrent la chaîne essaient d'envisager les situations les plus communes. Il est inconcevable qu'ils puissent construire des chaînes à la main en examinant chaque document source.

Notre objectif est donc l'amélioration continue de la chaîne de traitement, de sorte que le système apprenne de ses erreurs. Pour cela, on peut tirer parti du fait que des opérateurs humains consultent les fiches synthétiques produites par le système, qui contiennent les événements extraits et les pointeurs vers les documents sources. Ces utilisateurs ont la possibilité de corriger le contenu extrait, fournissant ainsi, indirectement, un *feedback* sur le traitement effectué.

À moyen terme, nous cherchons à développer un système qui réponde aux besoins réels de la communauté *Open-source intelligence* (OSINT). L'utilisateur corrigera les erreurs, et le système pourra prendre en compte ce *feedback* implicite sur les traitements qu'il a effectués, afin d'améliorer ces derniers pour les documents suivants.

Toutefois, dans cet article, nous nous intéressons à un objectif simplifié, dans lequel le *feedback* est supposé donné de façon explicite (simulé dans nos expériences), et sur une échelle numérique. Ce cadre simplifié est une première étape vers la résolution du problème, et nos expériences fournissent ainsi une preuve de concept pour notre objectif à moyen terme.

4.1 Formalisation comme un problème de décision

Puisqu'une chaîne unique pour traiter parfaitement chaque type de document est impossible à construire, l'idéal serait une chaîne faite sur mesure pour chaque document. Nous choisissons de modéliser le problème du traitement d'un document comme un processus de décision markovien (MDP). La stochasticité nous permet de prendre en compte, en particulier, le fait que des actions menées dans un contexte apparemment similaire, peuvent ne pas produire le même résultat. À titre d'exemple, choisir d'extraire la langue peut résulter en l'extraction de langues différentes, ce qui est pris en compte directement par les actions stochastiques des MDP.

Le système a une perception de la tâche sous la forme d'*états* du processus : document courant, informations déjà extraites, temps déjà passé sur ce document, etc. Chaque passage par un service modifie l'état courant. Par ailleurs, le système dispose d'un certain nombre d'*actions* qu'il peut appliquer dans l'état courant : ces actions correspondent au service suivant à lancer (ou à l'arrêt du traitement et l'enregistrement des événements extraits en base). La répétition d'une même action sur un même document est techniquement autorisée, mais cela sera pénalisé par le système de récompense (car induisant un temps de traitement plus long). La chaîne de traitement n'est plus figée, mais contrôlée par un algorithme de RL.

Les actions font transiter le système d'un état à l'autre : par exemple, l'action consistant à extraire la langue, appliquée dans un état donné s_t correspondant à un document en cours de traitement, fera transiter le système vers un état s_{t+1} égal à s_t à ceci près qu'il contiendra l'information « langue extraite » et l'annotation « fr ». Enfin, les récompenses $r(s)$ sont données au système en fonction du *feedback* sur les événements extraits (simulé dans notre cas), et donc seulement pour les états terminaux d'un traitement. Les volumes de documents traités en production (potentiellement tous les documents possibles du *web*) étant très importants, le temps passé à traiter le document influe également sur la récompense (*cf.* section 5).

Plus précisément, les états que perçoit la chaîne sont des états combinatoires, formés par les valeurs d'un certain nombre de *descripteurs* des documents. Ces états permettent une généralisation en apprentissage, par exemple, nous avons déjà vu que le *type* de document (politique vs. mariage) influence en grande mesure l'utilité du mot « alliance » pour l'extraction de l'information. On peut espérer que la chaîne apprenne au fil des interactions que

— si l'état courant a la valeur « true » pour le descripteur « typeExtrait » et la valeur « politique » pour le descripteur « type », alors la meilleure action à effectuer consiste à lancer un service d'extraction qui utilise « alliance » parmi les mots déclencheurs,
— dans les autres cas où le type est extrait, la meilleure action consiste à arrêter le traitement (inutile d'essayer d'extraire des accords entre pays dans des documents non politiques),
— sinon, la meilleure action consiste à lancer un service de reconnaissance de type.

5 Cadre expérimental

L'objectif de cet article est de donner une preuve de concept de notre approche. Pour cela, nous avons considéré un corpus de textes, dont les événements d'intérêt sont déjà connus. En appliquant notre formalisme, nous avons utilisé un algorithme de *Q-learning* pour contrôler le traitement, en simulant un *feedback* en utilisant la vérité terrain.

Nous nous basons sur une chaîne simple mais typique (*cf.* section 2). La chaîne est écrite comme une route Camel (2015) en XML. Chaque service est défini comme un *endpoint*, et nous utilisons le *Dynamic Router* pour donner à l'IA le contrôle sur les services appelées, leur ordonnancement, et leurs paramètres, spécifiquement, le choix des *gazetteers* (les mots déclencheurs) de GATE (Cunningham *et al.*, 2011) pour la détection des événements dans un texte.

5.1 Corpus

Nous nous intéressons à l'extraction d'événements correspondant à des attentats à la bombe (*bombings*) dans le monde entier. Le *Global Terrorism Database* (GTD (2014)) est une base

de données *open source* composée des détails de plus de 125 000 événements terroristes mondiaux de 1970 à 2013. Le corpus est formé d'un ensemble des synthèses de ces événements $\{d_1 \dots d_N\}$ d'où une chaîne d'extraction parfaite extrairait les événements $E_1, \dots, E_N \in GTD$, respectivement (*cf.* exemple 1, exemple 2). Nous attendons de notre système qu'il apprenne une chaîne qui s'approche de ce but, en apprenant non seulement le bon ordonnancement des services, mais aussi le fait que certains services (dans notre cas, le service d'inférence de l'information géographique) et certains *gazetteers* de GATE ne sont pas utiles.

5.2 États et actions

Un état est représenté par une affectation des caractéristiques : *language* $\in \{$"en","$\,$"$\}$, *format* $\in \{$"*text/plain*","$\,$"$\}$, *durée*, *nbServices* $\in \{0-5, 5-20, 20+\}$, *bombing* $\in \{$*true, false*$\}$, *any* $\in \{$*true, false*$\}$ où *durée* est le nombre de secondes écoulées depuis le début du traitement du document courant (arrondie à la dizaine de secondes), *nbServices* est le nombre de services déjà utilisés sur ce document, *bombing* est *true* si et seulement si un événement de ce type a déjà été extrait, et de même, *any* est *true* si et seulement si un événement quelconque a déjà été extrait. Ces caractéristiques sont choisies à titre illustratif pour la preuve de concept, en lien avec un ensemble restreint d'actions. Un système opérationnel prendrait évidemment en compte de nombreuses autres caractéristiques (type de document, liste complète de langues, etc.).

Les actions disponibles consistent à choisir le prochain service parmi {Tika, NGramJ, GATE, Geo}. Quand le système choisit GATE, il a le choix parmi six *gazetteers* : *bombing* (verbes et noms), *injure* (verbes et noms), *HarryPotter* (verbes et noms). Les mots contenus dans les listes de *bombing* et *injure* peuvent déclencher l'extraction des événements. Les listes *HarryPotter* contiennent exclusivement des mots qui ne sont pas présents dans les documents du GTD, tels que *dragon*, et l'IA devrait donc apprendre que ce paramètre du service est inutile. Nous cherchons ainsi à vérifier que, pour la tâche d'extraction de *bombings* qui lui est confiée, le système réussit bien à apprendre que le *gazetteer* le plus pertinent est *bombing*, et que les deux autres *gazetteers* l'induisent en erreur (si les mots d'*injure* sont utilisés pour détecter des *bombings*) ou le ralentissent inutilement (*gazetteer HarryPotter*). Une dernière action disponible permet au système d'arrêter le traitement du document, et de retourner les éventuels événements extraits.

5.3 Protocole

Nous avons pris deux jeux de documents du GTD qui contiennent de l'information sur des *bombings*. Le premier consiste en 100 documents d'entraînement que nous avons traités 30 fois par lots, utilisant un taux d'exploration ϵ de 0.4 qui est divisé par 2 jusqu'à 0.1 tous les 5 lots pour réduire de plus en plus l'exploration et augmenter l'exploitation. Les documents étant hétérogènes, nous avons utilisé un taux d'apprentissage α de 0.2. Cela ne garantit pas la convergence de l'apprentissage, mais permet de réduire l'effet négatif des grandes variations, par exemple, si plusieurs documents à la suite ne contiennent pas d'information extractible.

En contrôle, nous comparons la performance de deux chaînes « expertes » (*Mixte* qui accède à toutes les *gazetteers* de GATE, et *Bombe* qui n'accède qu'au *gazetteer* de *bombing*) à celle de l'IA sur le même jeu de 100 documents. Le deuxième jeu consiste en 1000 documents aléatoirement choisis que l'IA « voit » pour la première fois. Nous les traitons avec les deux chaînes « expertes » sans IA ; avec une IA paramétrée avec $\epsilon = \alpha = 0$ et les Q-valeurs apprises à l'issue

du lot 30 ; et avec une IA « vierge » avec un $\epsilon = 0.4$ initialement, divisé par 2 tous les 100 documents jusqu'à $\epsilon = 0.05$, et $\alpha = 0.2$. Pour éviter que l'IA tourne à l'infini, si le temps de traitement d'un document dépasse 30 secondes [1], la chaîne est arrêtée dès que le service courant a fini, induisant une récompense réduite ou négative, et les éventuels événements extraits sont retournés. Les documents sont toujours traités dans le même ordre.

5.4 Objectifs et récompenses

Le *feedback* donné au système prend en compte la similarité entre les événements potentiellement extraits par la chaîne sur le document et l'événement du GTD qui correspond à ce document (voir ci-dessous), et le temps passé à traiter le document. Précisément, en notant $Sim(E_e, E_b)$ la similarité moyenne entre les événements extraits (quand ils existent) et un *bombing* réel, et t le temps passé par la chaîne sur le document, nous donnons le *feedback* F suivant : Si $Sim(E_e, E_b) = 0$ (càd si $E_e = \emptyset$ ou $E_e \neq E_b$) alors F est $-t$, sinon, F est $\frac{Sim(E_e,E_b)}{\max(t,25)}$.

Nous formalisons ainsi le fait que l'extraction d'événements corrects est primordiale, et doit se faire dans un temps raisonnable. D'autre part, si aucun événement n'a été extrait, la chaîne est pénalisée, et ce d'autant plus que le temps passé est long. Nous encourageons ainsi la chaîne à détecter des événements, ou à détecter rapidement qu'il n'y a aucun événement d'intérêt dans le document.

Nous définissons la similarité entre deux événements $E_1 =< C_1, T_1, S_1, A_1 >$ et $E_2 =< C_2, T_2, S_2, A_2 >$ par $Sim(E_1, E_2) = (\alpha \cdot Sim(C_1, C_2) + \beta \cdot Sim(T_1, T_2) + \gamma \cdot Sim(S_1, S_2) + \delta \cdot Sim(A_1, A_2))/(\alpha + \beta + \gamma + \delta)$

Puisque nous simulons un utilisateur intéressé principalement par les événements de type *bombing*, nous donnons à α une valeur plus élevée, spécifiquement $\alpha = 20; \beta = \gamma = \delta = 1$.

La similarité conceptuelle $Sim(C_1, C_2)$ est de 1 s'il y a un atome commun entre les dimensions conceptuelles de E_1 et E_2, et de 0 sinon. Par exemple, pour $C_1 = \{BombingEvent, AttackEvent\}$ et $C_2 = \{BombingEvent\}$, on obtient $C_1 \cap C_2 = \{BombingEvent\}$, et donc $Sim(C_1, C_2) = 1$. Nous procédons de même pour la similarité géographique $Sim(S_1, S_2)$.

Pour la similarité temporelle $Sim(T_1, T_2)$, s'il y a au moins un élément en commun nous donnons une similarité de 1, mais si une comparaison directe ne donne pas un résultat, nous utilisons l'information dérivée (jour, mois, année, jour de la semaine). Par exemple, pour $T_1 = \{7\ October\ 1969\}$ et $T_2 = \{Tuesday\}$, l'intersection est vide, mais en notant que le 7 octobre 1969 a été un mardi, nous obtenons $Sim(T_1, T_2) = \frac{1}{7}$.

Enfin, pour la similarité entre les dimensions agentives, nous utilisons la distance de Levenshtein sur chaque paire d'agents a_1, a_2 pris dans A_1, A_2 (nombre minimal de caractères à supprimer, insérer ou remplacer pour passer d'une chaîne à l'autre), rendue « floue » ($FSLS(a_i, a_j)$) en considérant les sous-séquences de la chaîne principale (Ginstrom, 2015). Nous définissons $Sim(A_1, A_2)$ comme $\max\{FSLS(a_1, a_2) \mid a_1 \in A_1, a_2 \in A_2\}$, si elle est au-dessus d'un certain seuil θ, et 0 sinon (en pratique, $\theta = 0.45$ donne de bons résultats). Par exemple, pour $A_1 = \{Le\ Professeur\ Tournesol\ PhD\}$ et $A_2 = \{Tournessol\}$, on obtient

$Sim(A_1, A_2) = FSLS(\cancel{Le\ Professeur}\ Tournesol\ \cancel{PhD}, Tournessol) \overset{1\ suppression}{=} 1 - (\frac{1}{9}) = 0.89$

1. Ce seuil est mal calibré : il est trop haut pour que l'IA apprenne à s'arrêter aussi rapidement que la chaîne experte. En observant qu'une chaîne experte prend en moyenne 25 secondes pour traiter de documents similaires, nous avons ajouté une marge de 5 secondes pour permettre à l'IA d'apprendre qui s'est finalement avérée inutile.

Nous ne nous intéressons pas icià l'association d'entités telles que *François Hollande / le président*, mais la modularité du système permettrait de prendre en compte cette similarité facilement en s'appuyant sur des ressources adéquates.

6 Résultats expérimentaux

Les diagrammes $1a$ et $1b$ dans Figure 3 montrent les récompenses reçues (par document et par ordre décroissant des récompenses respectivement) par une IA, et par les deux chaînes « expertes » sur les 100 documents d'entraînement. Dans le diagramme $1a$, les documents où l'IA a fait au moins une exploration sont marqués sur les axes en haut et en bas, et seulement les courbes des lots $1, 10, 20, 30$ sont montrées (les autres étaient similaires).

La courbe *IA Lot30* et la courbe *IA Lot1* montrent comme attendu que plus l'IA traite de documents, plus elle s'améliore. La comparaison avec la courbe *Bombe* montre que l'IA est capable d'une performance proche de celle de la chaîne experte. Notons que les documents où l'IA reçoit de moins bonnes récompenses que la chaîne experte sont dues aux explorations (p.ex. document 13), ou à la mauvaise calibration du seuil de temps (document 5). Parfois l'IA surpasse la chaîne experte (document 87) qui se traduit par une récompense (réduite) pour l'extraction d'un événement autre que *bombing*.

Pour nous approcher du cas d'utilisation réel, nous avons ensuite traité 1000 documents inconnus, choisis aléatoirement. Les diagrammes $2a$ et $2b$ montrent les récompenses reçues (par document, et par ordre décroissant des récompenses respectivement) par l'*IA formée* sur les 100 documents mentionnés ci-dessus, par les deux chaînes « expertes » (*Bombe* et *Mixte*), et par l'*IA non formée* » (c'est-à-dire sans *a priori* sur les besoins de l'utilisateur ni sur les documents). Nous voyons que, sauf mauvaise calibration du seuil, la courbe *IA formée* suit exactement celle de la chaîne experte *Bombe*, montrant qu'elle a bien appris à n'extraire que des événements *bombing*. $2c$ montre la moyenne cumulative (lissée pour les valeurs négatives) des différences entre les récompenses reçues par les IAs (*formée* et *non formée*), et la chaîne experte *Bombe*. Nous voyons que l'IA *formée* est capable d'une performance constante, et que l'IA *non formée* se stabilise au bout d'environ 600 documents, et donc généralise bien en apprentissage. Il s'avère également que l'IA a bien appris l'ordonnancement de la chaîne. Par exemple, dans l'état correspondant à un document sans type ni langue extraits, 0 secondes et 2 services déjà écoulés, et aucun événement extrait, la meilleure action apprise est de passer le document au service Tika, et dans l'état correspondant à un document où la langue, le type, et au moins un événement sont extraits, quelles que soient les valeurs des autres attributs, l'IA arrête le traitement de ce document. L'IA a aussi appris à optimiser la chaîne pour ce type de document. Elle n'appelle pas le service *Geo* et n'utilise que la liste de verbes de *bombing*, comme espéré. Cependant, elle a trouvé une solution inattendue avec le choix des noms *injure*, ce qui reflète peut être l'importance relative donnée par GATE lui-même aux noms et verbes pour les extractions.

7 Conclusion et perspectives

Nous avons proposé une formalisation du problème de l'amélioration continue d'une chaîne de traitement de documents, visant à extraire des événements dans des documents provenant de sources ouvertes, ainsi qu'une solution basée sur l'apprentissage par renforcement.

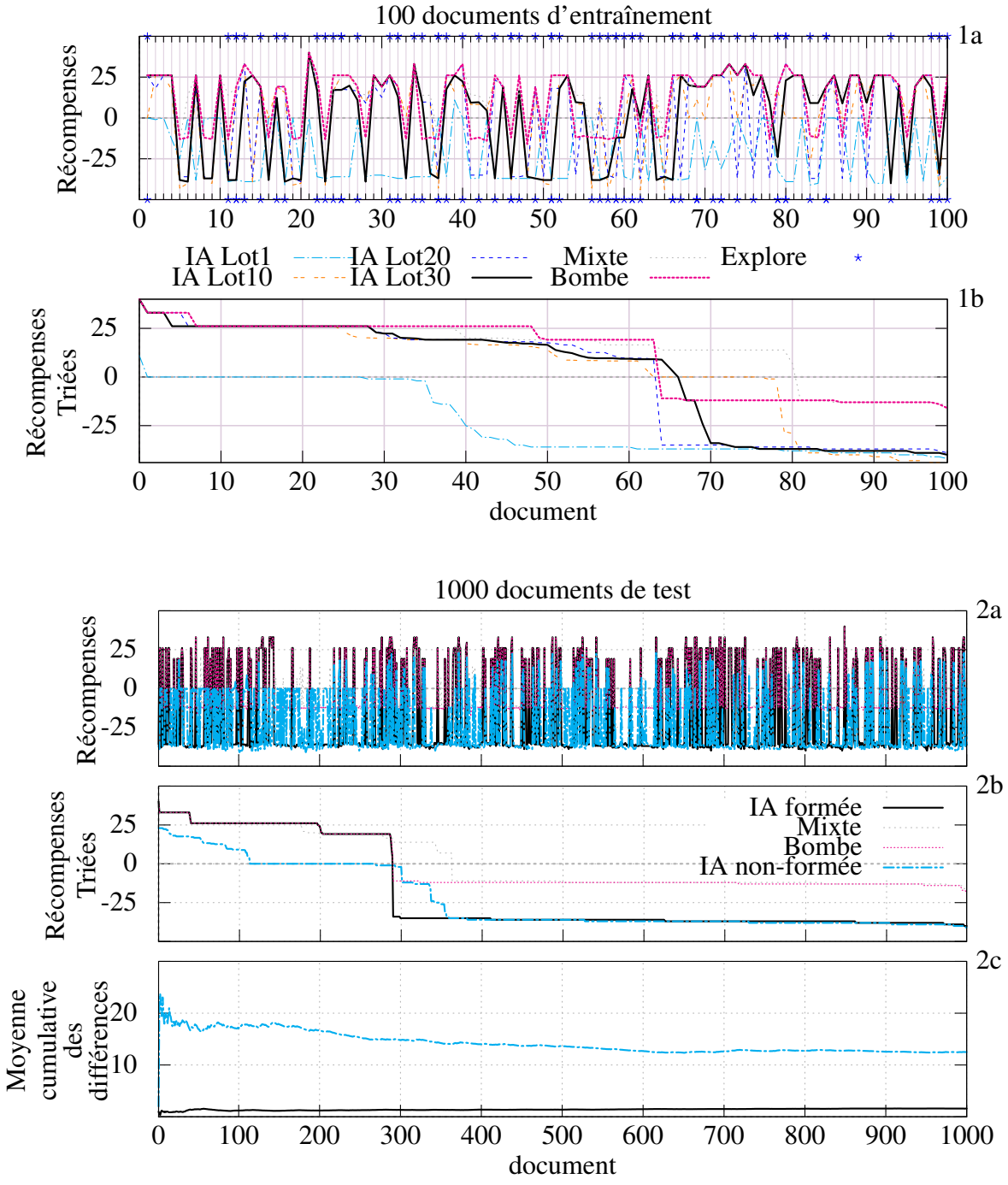


FIGURE 3 – Résultats expérimentaux : $1a, 1b$ montrent les résultats pour les documents d'entraînement, et $2a, 2b, 2c$ pour les documents de test ; $1a, 2a$ montrent les récompenses reçues par document (toujours rencontrés dans le même ordre), $1b, 2b$ montrent les récompenses triées, et $2c$ la moyenne cumulative lissée des différences entre le système experte et les IA.

Des premières expériences nous permettent d'apporter une preuve de concept à notre approche, et à court terme nous voulons nous approcher d'une situation de production en augmentant la taille des expériences, avec plusieurs types d'événements, une affectation plus complète de caractéristiques aux états, etc. Bien que nous ayons présenté ici un système simple, le temps de calcul ne sera pas un frein avec un algorithme de type *Q-learning*, dont les calculs sont instantanés à chaque étape.

Notre objectif final est d'intégrer le *feedback* implicite. Pour cela nous nous appuierons notamment sur Weng & Zanuttini (2013) et Weng *et al.* (2013), qui traitent de la prise en compte de récompenses qualitatives, de façon interactive, dans la prise de décision séquentielle.

Références

BRATKO I. & SUC D. (2003). Learning qualitative models. *AI magazine*, **24**(4), 107.

CAMEL (2015). Apache camel. `http://camel.apache.org/`.

CHAI X., VUONG B.-Q., DOAN A. & NAUGHTON J. F. (2009). Efficiently incorporating user feedback into information extraction and integration programs. In *Proc. SIGMOD 2009*, p. 87–100 : ACM.

CULOTTA A., KRISTJANSSON T., MCCALLUM A. & VIOLA P. (2006). Corrective feedback and persistent learning for information extraction. *Artif. Intell.*, **170**(14-15), 1101–1122.

CUNNINGHAM H., MAYNARD D., BONTCHEVA K., TABLAN V., ASWANI N., ROBERTS I., GORRELL G., FUNK A., ROBERTS A., DAMLJANOVIC D., HEITZ T., GREENWOOD M. A., SAGGION H., PETRAK J., LI Y. & PETERS W. (2011). *Text Processing with GATE (Version 6)*.

DOUCY J., ABDULRAB H., GIROUX P. & KOTOWICZ J.-P. (2008). Méthodologie pour l'orchestration sémantique de services dans le domaine de la fouille de documents multimédia.

DUPONT G., ADAM S. & LECOURTIER Y. (2011). Apprentissage par renforcement pour la recherche d'information interactive. In *Actes des JFPDA 2011*, p. Actes–électroniques.

FROMHERZ M. P., BOBROW D. G. & DE KLEER J. (2003). Model-based computing for design and control of reconfigurable systems. *AI magazine*, **24**(4), 120.

GEONAMES (2015). Geonames. `http://www.geonames.org/`.

GINSTROM R. (2015). The GITS Blog. `http://ginstrom.com/`.

GTD (2014). Global Terrorism Database. `http://www.start.umd.edu/gtd/`.

NGRAMJ (2015). Ngramj. `http://ngramj.sourceforge.net/`.

PUTERMAN M. L. (1994). *Markov Decision Processes : Discrete Stochastic Dynamic Programming*. New York : Wiley, first edition.

SERRANO L. (2014). *Vers une capitalisation des connaissances orientée utilisateur. Extraction et structuration automatiques de l'information issue de sources ouvertes*. Thèses, UNICAEN, France.

SUTTON R. S. & BARTO A. G. (1998). *Reinforcement Learning — An Introduction*. MIT Press.

SZEPESVÁRI C. (2010). *Algorithms for Reinforcement Learning*. Morgan and Claypool.

TIKA (2015). Apache tika. `http://tika.apache.org/`.

VAN HAGE W. R., MALAISÉ V., SEGERS R., HOLLINK L. & SCHREIBER G. (2011). Design and use of the Simple Event Model (SEM). *Web Semantics : Science, Services and Agents on the World Wide Web*, **9**(2), 128–136.

WATKINS C. J. C. H. (1989). *Learning from delayed rewards*. PhD thesis, University of Cambridge.

WEBLAB (2015). Weblab. `http://weblab-project.org/`.

WENG P., BUSA-FEKETE R. & HÜLLERMEIER E. (2013). Interactive Q-Learning with Ordinal Rewards and Unreliable Tutor. In *ECML/PKDD Workshop on RL with Generalized Feedback*.

WENG P. & ZANUTTINI B. (2013). Interactive Value Iteration for Markov Decision Processes with Unknown Rewards. In *International Joint Conference on Artificial Intelligence*.