Apprentissage incrémental de modèles de domaines par interaction dialogique

Vincent Letard

NNT : 2017SACLS100

THÈSE DE DOCTORAT
DE L'UNIVERSITÉ PARIS-SACLAY
PRÉPARÉE À L'UNIVERSITÉ PARIS-SUD

Ecole doctorale n°580
École doctorale de sciences et technologies
de l'information et de la communication
Spécialité de doctorat : Informatique
par

M. VINCENT LETARD

Apprentissage incrémental de modèles de domaines par interaction dialogique

Thèse présentée et soutenue au LIMSI, bâtiment 508, campus Paris-Sud, le 28 avril 2017.

Composition du Jury :

M.	PIERRE ZWEIGENBAUM	Directeur de recherche LIMSI CNRS	(Président du jury)
M.	PHILIPPE LANGLAIS	Professeur Université de Montréal - RALI	(Rapporteur)
Mme	ISABELLE TELLIER	Professeure Université Paris 3 Sorbonne Nouvelle - LaTTiCe	(Rapporteure)
M.	GILLES RICHARD	Professeur Université Paul Sabatier - IRIT	(Examinateur)
Mme	SOPHIE ROSSET	Directrice de recherche LIMSI CNRS	(Directrice de thèse)
M.	GABRIEL ILLOUZ	Maître de conférence Université Paris-Sud	(Co-encadrant de thèse)

Remerciements

La recherche est faite de quelques résultats spectaculaires pour de très nombreux tâtonnements, plus nombreux encore pour le doctorant, qui doit s'approprier son domaine de recherche en même temps que les méthodologies associées et les règles de la communauté scientifique. Mes premiers remerciements vont à mes merveilleux encadrants de thèse Sophie Rosset et Gabriel Illouz, dont j'ai pu bénéficier de l'expérience dans la recherche à tout point de vue. Je les remercie également pour leur confiance en mes choix scientifiques, parfois résolus, et leur soutien. Je leur dois les compétences et le recul que j'ai pu acquérir tout au long de ces quatre années. Pour tout cela, je leur suis profondément reconnaissant.

Je remercie bien sûr chacun des membres de mon jury de thèse, auxquels j'ai eu l'honneur de soumettre l'évaluation de ma thèse. Merci à Philippe Langlais, Isabelle Tellier, Gilles Richard et Pierre Zweigenbaum pour leur intérêt dans mon travail et leurs remarques extrêmement intéressantes. Tout particulièrement, je remercie M. Langlais pour nos échanges passionnants à l'occasion de la conférence ICCBR, notamment sur le présent et l'avenir de la recherche sur l'analogie formelle.

Les échanges sont un aspect important de la recherche scientifique, et à ce titre je souhaite remercier Olivier Galibert pour son aide experte dans le domaine de l'algorithmique, François Yvon pour m'avoir aiguillé dans le domaine au début de mon intérêt pour l'analogie, Thomas Lavergne pour ses explications patientes au sujet des modèles statistiques de traduction, Patrick Paroubek pour nos longues discussions déconstruisant les modèles pour mieux les repenser, ainsi qu'Yves Lepage pour nos échanges très intéressants sur l'analogie formelle.

Parallèlement à ma thèse, j'ai eu le plaisir d'enseigner l'informatique avec Anne Vilnat, Cécile Balkanski et Hélène Maynard. Je les remercie toutes trois également pour leur expérience, et leurs conseils sur ce monde voisin de celui de la recherche.

Je suis également très reconnaissant à Brigitte Grau qui m'a aidé et a souvent relu mon travail, et qui m'a permis de le terminer dans de bonnes conditions.

Le travail de thèse aurait été plus difficile sans l'environnement dont j'ai bénéficié au LIMSI, fertile aux discussions scientifiques, mais aussi épistémologiques, philosophiques, ou à propos de mille autres sujets parfois décalés. Je remercie pour cela chaleureusement tous les collègues, amis qui m'ont soutenu et avec qui les discussions ont toujours été aussi diverses qu'enrichissantes : Munshi, Marine, Guillaume, Leonardo, Swen, Julien, Arnaud, Mohamed, Damien, Jose, Pierre, Arthur, Mathieu-Henri, Romain, Korantin, Jean-Baptiste, Sami, Sanjay, Rashedur, Olivier, pour n'en citer que quelques uns...

Enfin, un grand merci à ma famille et mes amis pour leur soutien et leurs encouragements.

Table des matières

Introduction

J'ai des questions à toutes vos réponses.

Allan Stewart Konigsberg

Parmi les thèmes du domaine de l'intelligence artificielle, celui des assistants intelligents motive depuis une cinquantaine d'années les chercheurs à concevoir des modèles pour raffiner l'interaction humain-machine ainsi que l'apprentissage et le raisonnement artificiels. Les assistants intelligents ont également grandement inspiré la science-fiction et on les retrouve idéalisés dans des robots ou ordinateurs personnifiés tels que JARVIS, CASE/TARS, OS-One, C3PO, ou encore SIAAV [1]. Ils incarnent, ainsi que beaucoup d'autres, un ensemble de caractéristiques que la fiction prête aux systèmes intelligents. L'objectif de ce travail s'inscrit dans le domaine de la conception d'assistants intelligents capables d'interagir naturellement avec les humains. Il nous faut immédiatement préciser que l'ambition de ce travail n'est évidemment pas la réalisation d'un tel système, au comportement social complexe, sorti d'une imagination fertile. Nous entendons contribuer aux aspects les plus clairement définis de son comportement tels que que la réaction systématique à des instructions spécifiques dont l'évaluation est objective, en nous focalisant parmi eux sur les problèmes les plus centraux, c'est-à-dire les capacités de raisonnement et d'apprentissage. Ainsi, nous laissons de côté les aspects sociaux et émotionnels des agents intelligents, et nous restreignons les modalités d'interaction à l'écrit uniquement. Nous nous intéressons en revanche aux capacités des systèmes à effectuer des actions suivant les requêtes de l'humain, à apprendre de nouvelles tâches, à corriger leurs propres erreurs et à interagir par la langue naturelle. En particulier, la polyvalence nous semble être une caractéristique fondamentale pour un assistant intelligent, et nous fixons comme ligne directrice générale de ce travail l'objectif d'indépendance à la langue employée par l'humain, ainsi qu'à l'architecture de la machine permettant d'accomplir la tâche.

Parmi les capacités listées, celle qui nous semble être la plus importante est la capacité du système à apprendre. À aucun moment nous ne pouvons supposer tout connaître, c'est-à-dire être dans la situation dans laquelle aucun apprentissage n'est plus possible car aucune information n'est nouvelle. Si cela est vrai pour l'humain, nous admettrons *a priori* [2] que c'est le cas également pour un système intelligent. Plus précisément, de nouvelles situations peuvent survenir pour le système sous la forme de nouvelles manières de s'exprimer d'un utilisateur, ou encore d'une tâche inédite à réaliser. Ces deux cas peuvent provenir d'un changement dans l'expression ou les besoins de l'utilisateur du système, ou bien d'un changement d'utilisateur. Il faut donc envisager la capacité d'apprentissage du système pour les dépasser.

Plusieurs sources permettent d'acquérir de nouvelles connaissances : *l'expérimentation*, ou interaction avec l'environnement, *l'étude*, ou acquisition de connaissances encyclopédiques, et *l'enseignement*, ou interaction avec un professeur. Considérons chacune de ces sources dans le contexte d'un assistant intelligent.

L'*expérimentation* est une méthode systématique mais risquée, elle nécessite un environnement expérimental protégé, c'est-à-dire indépendant de l'environnement d'utilisation. Cette indépendance est parfois complexe à établir, et n'est pas toujours possible. Aussi est-il préférable d'éviter que le système mène des expérimentations de son propre chef afin de découvrir par exemple si la suppression irréversible d'un répertoire permet ou non de répondre à la demande de l'utilisateur.

L'*étude* comporte une large part de compréhension et d'interprétation des connaissances documentées afin de les ramener à un cas particulier et d'en extraire un savoir-faire. Cette problématique intéresse des champs d'étude tels que ce qu'on nomme en anglais le *Machine Reading*, ou encore la fouille de texte et l'extraction de connaissances. Cependant, suivant la remarque faite précédemment, nous

1. On aura reconnu les personnages des œuvres respectives : *Iron Man*, *Interstellar*, *Her*, *Star Wars*, ainsi que *La Ballade de Pern*.

2. La question de l'équivalence potentielle entre le cerveau humain et une machine apprenante relève de la philosophie, du moins tant que les connaissances sur la neurologie resteront parcellaires. Nous ne prétendons pas ici trancher cette question, mais seulement mettre en évidence la finitude de tout ensemble de connaissances : on considère qu'une exception est toujours susceptible de survenir.

ne pouvons supposer que les sources encyclopédiques contiennent l'intégralité des connaissances qui seront un jour nécessaires. Nous pouvons concéder tout au plus que les nouvelles connaissances y seront périodiquement ajoutées par des spécialistes du domaine correspondant, à l'image de Wikipedia. Sous cet angle, l'extraction d'informations depuis une base de connaissances qui requiert l'intervention d'un spécialiste semble redondante avec l'enseignement direct par ce spécialiste, à la différence que l'acquisition par extraction passe par un format intermédiaire d'expression des connaissances.

L'*enseignement* permet la transmission directe du professeur/spécialiste/expert à l'apprenant, ainsi que l'intégration de l'information sous une forme réutilisable par ce dernier. Cela implique, dans le cas où l'apprenant est un système artificiel, l'intégration de l'information sous une forme réutilisable par *tous*, car les données mémorisées peuvent être lues et copiées depuis le système informatique, contrairement au cas d'un "système biologique". Cette remarque ainsi que la possibilité d'une meilleure maîtrise de la périodicité des ajouts nécessaires aux connaissances du système – grâce à une interaction à la demande avec un ou plusieurs experts – motivent notre choix pour l'enseignement comme source simple, fiable, dynamique et sûre de connaissances pour un assistant intelligent. Cela suppose bien entendu que l'enseignant sollicité est lui-même tout à fait fiable et n'enseigne que des connaissances correctes (admises). Nous adoptons cette hypothèse, mais ses conséquences en termes de performances et d'éthique [3] ne doivent pas être ignorées pour autant. En effet, sous cette hypothèse, le système peut bénéficier d'une confiance excessive de la part des utilisateurs alors que l'absence d'erreurs dans la base de connaissances ne prévient pas de l'absence d'erreurs dans les choix du système. De plus des problèmes éthiques peuvent se poser si l'on considère le risque de l'enseignement de connaissances erronées par mégarde ou par malveillance, d'autant plus si le système a vocation à partager ses capacités pour de multiples utilisateurs.

Nous considérons donc un assistant intelligent apprenant de manière incrémentale auprès d'utilisateurs experts. En particulier, nous nous intéressons au cas dans lequel les connaissances initiales de ce système sont limitées ou nulles. On peut comparer cette situation, comme le proposait Turing [Turing, 1950], à celle d'un enfant dont les connaissances sont d'abord nulles, mais dont les capacités innées lui permettent d'apprendre à partir de son environnement pour enrichir ou corriger/affiner ses connaissances [Tellier, 2005]. Notre approche cependant, ne visera pas à modéliser l'apprentissage pour l'exhaustivité des domaines auxquels est confronté un enfant, il n'est pas question notamment, de considérer l'apprentissage de l'expression en langue naturelle. Notre étude est focalisée sur l'apprentissage de modèles d'un domaine spécifique, il s'agit en l'occurrence de l'amélioration des actions (non langagières) effectuées par le système assistant en réponse aux requêtes de ses utilisateurs.

L'usage d'un assistant apprenant aux connaissances initialement limitées conserve néanmoins son intérêt pour l'utilisateur expert pour plusieurs raisons. L'utilisateur peut s'appuyer sur l'assistant intelligent comme aide mémoire pour les tâches inhabituelles, ou bien comme aide cognitif capable par son raisonnement de réaliser des tâches complexes dont l'utilisateur n'a qu'une connaissance latente. L'intérêt de l'usage d'un tel assistant pour un utilisateur novice vient quant à lui une fois que le système atteint une quantité de connaissances et une performance satisfaisantes.

Afin de suivre l'objectif de généricité que nous nous sommes fixé, le format des connaissances doit convenir quelle que soit la méthode utilisée pour la production d'actions à partir des requêtes soumises par l'utilisateur. Une représentation générique d'une association entre une prémisse en langue naturelle et un résultat attendu doit minimiser les traitements appliqués avant sa mémorisation. Nous utilisons des associations entre chaînes de caractères en langue naturelle et séquences d'instructions à effectuer. L'expression de cette séquence d'instructions est sous la forme d'une chaîne de caractères en langage formel, permettant à la machine d'exécuter les actions de manière systématique. La tâche du système

3. cerna-ethics-allistene.org

apprenant est donc représentée comme le transfert d'une chaîne de caractères en langue naturelle vers une chaîne de caractères en langage formel [4].

On dispose ainsi d'une base de connaissances sous forme d'associations, que l'assistant intelligent peut utiliser pour réagir aux requêtes des utilisateurs. L'étape suivante est de concevoir une méthode pour combiner ces connaissances afin que l'assistant soit capable de couvrir plus que les seules requêtes déjà présentes dans la base. Nous proposons d'utiliser le raisonnement par analogie formelle. Il s'agit d'une formalisation du raisonnement analogique, aussi appelé règle de trois ou quatrième de proportion, tel que l'humain l'applique très fréquemment [Hofstadter et Sander, 2013]. Son principe est celui du raisonnement à base de cas, par lequel un problème est résolu à l'aide d'un exemple connu, allant du particulier au particulier. Une analogie s'établit entre quatre objets et s'énonce : "A est à B ce que C est à D". On peut ainsi prédire l'objet D grâce à aux relations respectives de A avec B et de A avec C. Le raisonnement par analogie formelle permet, nous le verrons, de produire des réponses pertinentes à partir d'un ensemble d'exemples restreint. Il est également intelligible en tant que dérivé d'un mode de raisonnement qui nous est familier. Ce sont les deux raisons principales qui ont motivé notre choix de l'analogie pour opérer le transfert entre langue naturelle et langage formel.

Les contributions de ce travail sont de plusieurs ordres. En premier lieu, et afin d'évaluer les modèles proposés, nous avons collecté des ensembles d'exemples associant des requêtes en langue naturelle (français) à des commandes en langage formel (`R` et `bash`). Nous avons également étudié les biais possibles liés à la collecte de ce type d'associations, et essayé différents protocoles afin d'en limiter les plus indésirables.

D'autre part, nous avons proposé différentes méthodes pour le transfert depuis une langue naturelle vers un langage formel. Nous avons développé une méthode fondée sur la similarité entre requêtes, utile en tant que référence pour les méthodes fondées sur l'analogie. Concernant ces dernières, nous avons proposé dans un premier temps l'application simultanée de deux types de résolution en soulignant son intérêt dans le cas précis du transfert entre langue naturelle et langage formel. Nous avons également considéré l'extension de la base d'exemples par la génération analogique systématique pour en étudier l'impact sur les réponses proposées par le système. Dans un second temps et au vu du taux de réponse relativement bas relevé dans les expériences précédentes, nous avons proposé de relâcher les contraintes analogiques afin de limiter la sensibilité au bruit trouvé dans les expressions en langue naturelle. Enfin nous proposons un protocole d'étude de la capacité d'apprentissage du système par l'incrément de ses connaissances, simulant l'interaction avec un utilisateur expert à chaque erreur constatée. Nous avons étudié les courbes d'apprentissage produites à partir d'une base de connaissances initialement vide, et également à partir d'une base de taille fixée, en soumettant des requêtes d'un nouvel utilisateur du système.

Le document est organisé en deux parties, couvrant sept chapitres. La première partie constitue une introduction détaillée aux diverses problématiques mises en jeu, à l'issue de laquelle nous apportons un nouveau regard sur le sujet. Nous présentons dans le premier chapitre un état de l'art sur le transfert de langage, les systèmes interactifs et apprenants. Le chapitre 2 détaille les collectes d'exemples que nous avons effectuées, et analyse les différents biais inhérents à chacune d'elles. Nous rapportons dans le chapitre 3 les résultats obtenus à l'aide d'une approche de transfert fondée sur la similarité avec les requêtes de la base. Ce chapitre est également l'occasion de noter des limites fortes liées à ce type d'approche pour le cas du transfert d'une langue naturelle vers un langage formel. Nous justifions alors l'intérêt de l'analogie pour les dépasser, introduisant la seconde partie. Le chapitre 4

4. Nous utilisons dans ce document le terme de "langue" pour désigner une langue naturelle, et celui de "langage" pour un langage formel afin de les distinguer. L'acception de "langage" tel que nous l'employons relève donc de la théorie des langages et non de la psychologie sociale.

est une introduction détaillée du raisonnement par analogie formelle, des algorithmes associés de l'état de l'art, ainsi que de leurs limites computationnelles. Nous présentons ensuite l'usage dual que nous avons proposé de l'analogie formelle pour le transfert direct et indirect dans le chapitre 5. Le chapitre 6 décrit ensuite notre proposition visant à réduire la sensibilité du raisonnement par analogie formelle au bruit présent en entrée par la relaxation de contraintes. Enfin, le chapitre 7 reprend l'approche proposée au chapitre 5 pour l'appliquer dans un contexte incrémental simulé et étudier avec plus de précision le comportement du système au cours de l'apprentissage. Nous résumons en conclusion les résultats obtenus tout au long de nos travaux et leurs implications, avant d'ouvrir sur les nombreuses perspectives qui en découlent tant sur le plan scientifique qu'applicatif.

Première partie

Objectifs et positionnement

Chapitre 1

Contexte

Sommaire

1.1 Introduction

Pour considérer la question de l'apprentissage par l'interaction, nous nous intéressons d'abord à la définition de l'apprentissage.

Le verbe apprendre de la langue française dérive étymologiquement du latin "apprehendere", "apprehendo" : préfixes "ad-" (vers), "pre-" (avant) et radical "hendo" du grec ancien χανδάνω (contenir). À titre de comparaison, le mot anglais "learning" quant à lui provient de l'ancien anglais "leornian", lui-même de racine proto-germanique "liznaną", "lizaną", remontant au proto-indo-européen "leys" dont le sens est celui de suivre, sillonner, pister. Le terme d'apprentissage est très polysémique donc. Il peut désigner aujourd'hui l'acquisition ou la révélation d'une information (*apprendre une nouvelle*), l'acquisition d'un savoir faire par la pratique physique (*apprendre à conduire, à nager*), ou intellectuelle (*apprendre à dériver une fonction, à programmer*), la mémorisation consciente d'une plus grande quantité d'informations, ou d'un motif complexe (*apprendre les tables de multiplications, un discours*), ou encore l'adoption d'un ensemble de valeurs comportementales (*apprendre les bonnes manières*). Les sens de certaines de ces acceptions se recoupent, mais les concepts qu'elles mettent en jeu sont distincts et tous ne sont pas immédiatement pertinents lorsqu'on parle d'intelligence artificielle. En effet les deux dernières acceptions listées du terme "apprendre" ne s'appliquent pas à une machine, du fait de son fonctionnement différent de la mémoire biologique et parce les objectifs qu'on lui fixe ne sont pas les mêmes que ceux de l'humain. Ces questions restent hors de notre propos ici.

La mémorisation et restitution de connaissances "par cœur" ne pose pas de problème en informatique tant qu'il ne s'agit pas de raisonner sur ces connaissances. L'apprentissage de savoir-faire ou compétences, c'est-à-dire de la capacité d'adapter et de combiner les connaissances mémorisées, est en revanche plus problématique. C'est l'objet de la discipline de l'apprentissage artificiel. Il relève de la deuxième et de la troisième des acceptions listées plus haut. Nous nous intéressons en particulier à la troisième, qui concerne la réalisation d'un objectif requérant la maîtrise d'un savoir ou d'une technique (sans mettre en jeu de capacité motrice). Cet objectif pourra être atteint, ou mieux approché, par la restitution ou la réutilisation de connaissances acquises. En intelligence artificielle, on désigne par apprentissage (*machine learning* en anglais) les méthodes permettant de construire un modèle prédictif d'un phénomène à partir de ses manifestations, ou exemples. L'exploitation de ce modèle pour traiter de nouveaux exemples est donc appelée prédiction.

L'**apprentissage artificiel**[*] est la discipline de l'étude et de la recherche des méthodes d'acquisition automatique de connaissances et de leur représentation [Cornuéjols et Miclet, 2011]. Puisque l'on assimile une connaissance à une compétence, il s'agit alors de formaliser l'acquisition par le système de la capacité d'effectuer des tâches. Les tâches formalisées sont les tâches qui peuvent être exécutées de manière systématique, notamment par une machine. L'exécution systématique d'une tâche requiert de pouvoir la spécifier à l'aide d'un langage formel.

Nous rendons compte dans la section 1.2 de la tendance en informatique de rendre la spécification d'instructions de plus en plus naturelle. Cette tendance suppose la capacité de transformer une spécification naturelle en une spécification dans un langage formel. En section 1.3, nous présentons un état de l'art des méthodes utilisées pour faire cette transformation. En vue de rendre l'interaction avec la machine plus naturelle, nous explorons en section 1.4 les techniques de gestion de dialogue de la littérature. Enfin, nous proposons en section 1.5 un modèle interactif minimal indispensable pour l'apprentissage incrémental, après avoir présenté une approche voisine existante.

1.2 Automatisation

L'histoire de l'informatique est fondamentalement liée à la systématisation de la résolution de problèmes en vue de leur automatisation. Il s'agit de faire effectuer par la machine des tâches fastidieuses

car répétitives ou difficiles car complexes. Ces tâches sont représentées pour la machine sous forme d'une suite d'instructions, exprimées à l'aide d'un langage formel de spécification donné, qu'on appelle **langage de programmation**[*]. Les langages de programmation ont connu un fort développement depuis les premiers ordinateurs, bien que quelques uns aient été conçus avant même leur apparition, comme les diagrammes d'Ada Lovelace pour programmer la machine de Charles Babbage. Depuis lors, de nombreux travaux n'ont cessé de rapprocher ces langages de spécification de la langue naturelle. La langue naturelle est le langage utilisé par l'humain, on le distingue du langage formel par la possibilité d'utiliser l'ambiguïté. Une expression est dite ambigüe lorsqu'elle peut référer à plusieurs sens comme c'est le cas en français du mot "plan" considéré hors de son contexte par exemple. Le langage formel n'est pas ambigu, chacune des expressions est associée à une unique sémantique, ce qui n'est pas le cas de la langue naturelle comme nous venons de le voir. Cette ambiguïté semble suivre la tendance du principe d'économie des signes (*cf.* maximes de Grice [Grice, 1975], et la théorie de la pertinence [Ducrot, 1998]) lui-même lié à la loi du moindre effort (ou principe d'économie) [Zipf, 1935], appliquée en particulier à la phonétique [Martinet, 1956]. En effet, la réduction du nombre de signes nécessaires à la communication d'un message cause, à partir d'un certain seuil, des collisions dans la relation des signes aux sens. Le sens est alors déterminé grâce à la réutilisation d'informations partagées : le contexte de la communication. À titre d'illustration, ce chapitre lui-même vise à établir un contexte partagé de communication grâce auquel il sera possible de développer les suivants, sans nécessité de rappeler les références introduites ici en extension.

On peut donc comprendre cette tendance à rapprocher les langages de spécification de la langue naturelle comme une conséquence de la loi du moindre effort appliquée à la spécification de processus de traitement de l'information.

En premier lieu, le nombre de signes d'une instruction ou d'un programme peut être réduit grâce à la réutilisation d'information contenue dans le programme lui-même. Il s'agit de la factorisation. C'est un principe fondamental en programmation, appelé *définition de fonction* : il consiste à donner un nom à une séquence d'instructions en vue de réutiliser cette séquence à l'aide d'un seul signe. La factorisation survient bien entendu dans les langues naturelles également avec l'apparition de nouveaux mots ou expressions, *i.e.* de nouveaux usages (exemple : le mot *laser*, issu d'un acronyme anglophone décrivant son principe). En programmation, les efforts de factorisation au niveau de la **compilation**[*] [1] ont donné lieu à l'apparition de langages de plus en plus expressifs dits de haut niveau, avec par exemple l'introduction d'opérateurs complexes, la gestion d'exceptions. Ces évolutions ont notamment permis l'apparition de nouveaux paradigmes de programmation, dont la programmation objet, logique, fonctionnelle, orientée-agent sont des exemples. L'utilisation de langages de haut-niveau permet d'accélérer et de simplifier le développement de programmes grâce principalement aux propriétés suivantes permises par la compilation et décrites dans [Mogensen, 2009] :

- la notation des langages de haut-niveau est plus proche de la manière dont les humains pensent les problèmes que celle du langage machine [2] ;
- le compilateur peut identifier et signaler des erreurs évidentes du programmeur ;
- les programmes écrits dans des langages de haut niveau sont généralement plus courts que leur équivalent en langage machine.

1. Compilation : opération de transcription d'un programme écrit dans un langage de haut niveau dans un langage de plus bas niveau.

2. Le langage machine est un langage formel représentant les instructions telles qu'elles sont exécutées par le processeur. Ces instructions ainsi que tous les paramètres sont donc encodés en binaire.

The distance from New York to Los Angeles is 3000 miles. If the average speed of a jet plane is 600 miles per hour, find the time it takes to travel from New York to Los Angeles by jet.

The price of a radio is 69.70 dollars. If this price is 15 percent less than the marked price, find the marked price.

EXEMPLE 1.1 – Problèmes résolus par le système STUDENT

```
1  Define a simple_report procedure UNIT_REPAIR_REPORT.
2  Fill the enumerator with a chain_enumeration of UNITS and REPAIRS.
3  Fill the main_file_key with a query_user_for_key of UNITS.
4  Fill the title with ("Report of Repairs on Unit" & UNIT_KEY).
5  Remove the summary.
```

EXEMPLE 1.2 – Un bloc d'instructions soumis au système KBEMACS

On peut également ajouter que de nombreuses erreurs de copie ou d'inattention disparaissent grâce à la factorisation. Ces mêmes arguments, à l'exception du second, motivent l'utilisation de la langue naturelle en tant que langage de spécification.

Le champ de la programmation automatique rassemble les efforts dans le but de rapprocher le langage de spécification pour les instructions données au système de la langue naturelle. Des approches pour cet objectif ont été proposées dès les débuts de l'informatique.

[Bobrow, 1964] décrit dans sa thèse le système STUDENT de compréhension et de résolution de problèmes d'algèbre simples formulés dans un anglais contraint, tel que présenté dans l'exemple 1.1, grâce à l'association de phrases avec des équations.

Ce mode d'expression des problèmes contraint l'ensemble des tâches traitées par le système. En outre, l'approche à base de règles proposée ne permet pas une application directe à d'autres modes d'expression ou d'autres langues naturelles.

La programmation est définie par Rich et Waters par : "l'expression d'un ensemble d'instructions dans un langage formel spécifique en vue de son exécution non ambigüe (généralement par une machine)". Afin de proposer une perspective unifiée sur la programmation automatique pour le projet *the Programmer's Apprentice*, Rich et Waters prennent pour modèle les choix de programmeurs experts vis-à-vis de tâches à accomplir [Rich et Waters, 1993]. Ils décrivent pour cela comment le programmeur analyse, synthétise, modifie, explique, spécifie, vérifie et documente les programmes. L'approche proposée est un **système d'aide à la programmation**[*] qui repose sur un module central de spécification. Ce dernier a pour tâche de transformer les descriptions informelles en spécifications formelles à l'aide d'une ontologie de notions prédéfinies (appelées *clichés*) qu'il identifie aux descriptions de l'utilisateur.

Le système KBEMACS (*Knowledge-Based Editor in Emacs*) transcrit des listes d'instructions en langue naturelle en un programme complet en langage ADA.

L'anglais utilisé pour les instructions emploie un vocabulaire spécifique de l'entreprise, relatif au traitement des informations de maintenance de machines (UNIT). L'exemple 1.2 illustre un bloc d'instructions tel que KBEMACS les traite. La base de connaissances est renseignée au préalable avec les concepts correspondants ainsi que leurs propriétés et relations. L'originalité de cette approche est sa généricité : il est possible de rendre le système compatible avec l'expression de besoins quelconques pourvu que l'anglais utilisé soit suffisamment simple et que le système dispose de la base de connais-

- *Put the pallet on the truck*
- *Go to the pallet on the truck*
- *Go forward and drop the pallets to the right of the first set of tires*
- *Pick up the pallet of boxes in the middle and place them on the trailer to the left*

EXEMPLE 1.3 – Instructions soumises au chariot élévateur robotisé

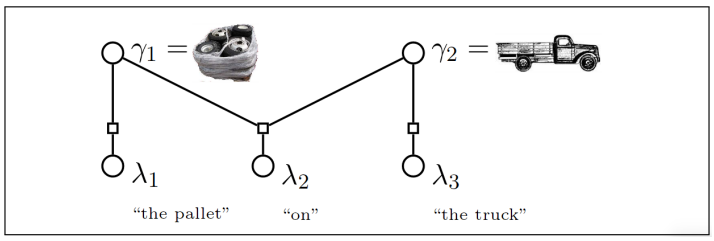


FIGURE 1.1 – Graphe d'ancrage généralisé associant les éléments linguistiques de l'expression "*the pallet on the truck*" aux représentations des objets "pallet" et "truck".

sances et du vocabulaire nécessaires. Cependant, cette adaptation requiert toujours l'intervention d'un expert pour la constitution de la nouvelle base de connaissances.

Le principe de réponse à des instructions formulées en langue naturelle a également été appliqué en robotique. On peut citer [Tellex *et al.*, 2011] qui propose une approche pour l'association d'instructions en langue naturelle, telles que lisibles dans l'exemple 1.3, à des séquences d'actions en vue d'exécuter l'instruction donnée.

À cette fin, leur approche s'appuie sur la structure linguistique des requêtes pour former un plan d'action pour le robot sous forme d'un graphe d'ancrage généralisé (*Generalized Grounding Graph*) représentant l'objectif spécifié. Un exemple de graphe d'ancrage généralisé est donné dans la figure 1.1, extraite de [Kollar *et al.*, 2013]. Ces graphes représentent une association entre des segments linguistiques de l'instruction et des objets, chemins, lieux et évènements de l'environnement du robot à l'aide d'une hiérarchie de clauses de description spatiales (*Spatial Description Clauses*). Ces clauses sont extraites à l'aide de l'analyse en dépendances produite par le Stanford Parser [De Marneffe *et al.*, 2006]. Un modèle probabiliste est appris pour l'association entre clauses et objets par apprentissage supervisé fondé sur les CRF (*Conditional Random Fields*) [Lafferty *et al.*, 2001]. Cette approche est ainsi applicable à tout type d'actions pourvu qu'on dispose d'exemples d'apprentissage qui les couvrent. Cependant, l'utilisation du Stanford Parser rend l'analyse du langage problématique car il s'agit d'un système spécifique à l'anglais [3]. La généricité ne s'étend donc pas à toute langue naturelle pour l'expression des instructions. D'autre part l'approche ne permet pas de traiter correctement les mentions d'objets n'ayant pas d'ancrage physique tels que "l'espace vide entre les deux piles".

Nous proposons de regrouper les systèmes de programmation automatique au sens large sous la dénomination d'**assistants opérationnels**[*]. Un assistant opérationnel est un système capable de comprendre des spécifications de haut niveau, en langue naturelle par exemple, et d'effectuer de manière autonome la tâche correspondante.

Dans la section suivante, nous nous intéressons aux différentes méthodes envisageables pour effec-

3. Même appliqué à l'anglais seul, le stanford parser n'est pas tout à fait adapté car entraîné sur des données composées principalement d'anglais journalistique (*Wall Street Journal*), biomédical (*Genia Treebank*) ainsi que plus courant (*Treebanks* de traductions anglais-arabe et anglais-chinois, et *QuestionBank*). Les ensembles d'entraînement ne comportent pas d'interaction humain-machine.

tuer le passage depuis la requête en langue naturelle vers une séquence d'actions appropriée à effectuer par le système. Dans un but de simplicité, nous proposons de considérer la séquence d'actions en tant que son expresion dans un langage de programmation. Cela permet de considérer le problème comme une tâche de transfert entre langages.

1.3 Transfert d'un langage vers un autre

Nous distinguons dans ce document les notions de traduction et de transfert. Le transfert d'une langue naturelle vers une autre est appelé traduction, automatique lorsqu'elle est effectuée par un système. Nous utilisons le terme de traduction pour désigner le transfert d'une langue naturelle vers une autre langue naturelle. Notons qu'on utiliserait également le terme de traduction pour désigner le transfert depuis un langage formel vers un autre langage formel (qu'on appelle dans certains cas compilation, *cf.* section 1.2), mais nous ne développons pas ce cas de figure et nous n'employons donc pas cette acception. Le terme de transfert lui-même désigne le passage d'un domaine quelconque à un autre domaine quelconque sans autre contrainte. Cependant nous l'utilisons essentiellement pour désigner le passage d'une langue naturelle à un langage formel, par contraste avec la notion de traduction.

Nous étudierons dans un premier temps les approches existantes en traduction automatique, avant de considérer des travaux plus spécifiques au transfert depuis une langue naturelle vers un langage formel. Dans un troisième temps, nous nous intéressons à un paradigme différent pour considérer ce transfert de langages : la recherche d'informations.

1.3.1 Traduction automatique

La traduction automatique relève habituellement du transfert entre deux langues naturelles et non entre une langue naturelle et un langage formel. Dans quelle mesure les méthodes employées en traduction automatique sont-elles généralisables au transfert vers un langage formel ?

Les premières techniques de traduction automatique étaient fondées sur des systèmes de règles de traduction, que l'on pourrait comparer aux compilateurs dont nous avons discuté plus haut. Un autre type d'approche, qui a connu un grand succès depuis, consiste à traduire les phrases à partir d'exemples de traductions déjà connus. Le succès de ce type d'approche repose sur l'étude du langage selon une démarche descriptive. Cette démarche, opposée à la démarche prescriptive, implique l'observation des usages de la langue en tant que tels, sans jugement de valeur ou catégorisation normative. Elle rend ainsi compte de la fluidité de la langue naturelle à travers le temps et les distances. En particulier, ce qu'on appellerait une "faute" dans une perspective normative, n'est alors considérée que comme un usage particulier de la langue. Cela permet dans certains cas de définir jusqu'à des idiolectes (dialecte au niveau de l'individu) ; nous y reviendrons dans le chapitre 2.

L'une des grandes approches pour la traduction automatique à base d'exemples est fondée sur l'utilisation de modèles statistiques des langues [Brown *et al.*, 1993]. On peut également la placer dans la catégorie des approches à base de cas puisqu'elle s'appuie sur de nombreux exemples de traduction pour construire un modèle.

L'approche statistique présente un inconvénient majeur vis-à-vis du transfert vers un langage formel : le besoin d'une grande quantité de données d'entraînement. Comme nous le verrons dans le chapitre 2, les bases d'association entre langue naturelle et langage formel sont rares et hétérogènes. D'autre part, les méthodes de traduction reposent sur la constitution d'un modèle de traduction de n-grammes (succession de n mots dans une phrase). En d'autres termes, chacun des n-grammes ($n \geq 1$) rencontrés en langue source est associé avec un m-gramme ($m \geq 0$) en langue cible. Ce type d'association ne permet pas de générer en sortie des segments discontinus tels que des expressions bien parenthésées d'un langage formel.

Affiche les noms et ville des vendeurs de la ville de Paris.	`SELECT nom, ville`
	`FROM vendeurs`
	`WHERE ville='Paris';`

Donne toutes les informations des clients qui ont été enregistrés avant 2014.	`SELECT *`
	`FROM clients`
	`WHERE inscription < '2014';`

EXEMPLE 1.4 – Exemple de correspondances entre requêtes en langue naturelle et requêtes en SQL.

Un autre type d'approche pour la traduction automatique à base d'exemples a été proposé dans [Nagao, 1984]. Nagao utilise la notion d'analogie pour identifier les associations entre des segments de la phrase en langue source et des segments de la phrase en langue cible. L'approche analogique a été reprise par la suite et progressivement formalisée, avec [Lepage et Denoual, 2005, Langlais et Patry, 2007] et d'autres travaux plus récents qui seront détaillés lors du chapitre 4.

Les méthodes de traduction par raisonnement à partir de cas n'impliquent pas *a priori* le même besoin de quantités de données que l'approche statistique. D'autre part, contrairement aux modèles de langage statistiques, la traduction à partir d'un seul ou d'un faible nombre d'exemples semble plus adaptée pour conserver la structure syntaxique stricte d'un langage formel en tant que langue cible.

1.3.2 Transfert depuis la langue naturelle vers le langage formel

Certains travaux ont déjà porté sur le transfert depuis la langue naturelle vers un langage formel. Parmi les applications les plus directes du transfert depuis la langue naturelle vers un langage formel, on trouve la famille de travaux portant sur la conception d'interfaces en langue naturelle avec des bases de données [Copestake et Jones, 1990, Androutsopoulos *et al.*, 1995, Popescu *et al.*, 2003]. Il s'agit de transférer une requête, souvent en anglais, en une instruction équivalente dans le langage de requête du système de gestion de la base de données (couramment SQL, ou NoSQL), *cf.* l'exemple 1.4.

Malgré la formulation simple de la tâche, la mise en place d'interfaces en langue naturelle soulève plusieurs problèmes [Kaufmann et Bernstein, 2007] :

- l'expressivité et l'ambiguïté de la langue naturelle sont complexes à prendre en compte, et le choix est fait en général de restreindre le langage de requêtes de l'utilisateur pour circonscrire les phénomènes à contrôler ;
- les interfaces efficaces sont souvent celles qui sont développées pour un domaine ou une application spécifique, elles sont donc d'autant moins adaptables à un autre domaine quelconque ;
- la performance des interfaces est directement liée à la spécialisation de son implémentation pour le domaine sur lequel elle est évaluée, cependant l'objectif est de concevoir des interfaces génériques en sacrifiant le minimum de performance ;
- l'utilité des interfaces en langue naturelle est également en question à une époque où les utilisateurs sont habitués à formuler leurs requêtes avec des mots-clés dans un moteur de recherche et à sélectionner de multiples résultats pertinents parmi la liste proposée.

Concernant le dernier point à propos de l'utilité des interfaces, Kaufmann et Bernstein ont observé que la modalité des requêtes en langage naturel est significativement préférée aux autres (mots-clés, navigation de menus, langage de requêtes graphiques) dans le cadre d'une interaction pour l'interrogation d'une base de données.

khalkis, Greece is a city	`(#$isa khalkis, Greece)`
the occupation of every pro deejay is deejay	`(#$relationAllInstance #$occupation (#$ProfessionalFn #$DiscJockey) #$DiscJockey)`
pronoun is one part of speech form for the word "hers"	`(#$partOfSpeech (#$T2-TheWord form part of speech) #$PossessivePronoun-Post)`
the latitude of Mashad is 36.27 degrees	`(#$latitude #$CityOfMashadIran (#$Degree-UnitOfAngularMeasure 36.27))`
lacrimal fluid is a type of body secretion	`(#$genIs #$LacrimalFluid #$Secretion-Bodily)`

EXEMPLE 1.5 – Exemple de traductions depuis l'anglais vers le langage CycL

Une famille d'approches pour les assistants opérationnels a émergé de l'utilisation des grammaires catégorielles combinatoires (GCC). Il s'agit d'un formalisme fondé sur la décomposition en constituants d'énoncés de la langue naturelle. Il s'appuie sur la logique combinatoire pour rendre compte de la structure des énoncés. Parmi les travaux utilisant ce formalisme, on compte [Vadas et Curran, 2005, Kushman et Barzilay, 2013]. Les GCC permettent de construire des règles de transformation pour le transfert depuis la langue naturelle vers un langage formel.

Une approche de traduction depuis l'anglais vers le langage formalisé CycL [4] a été implémentée dans le cadre du projet XLike [Tadić *et al.*, 2012]. L'objectif du projet est l'extraction de connaissances structurées à partir de documents multilingues. L'une de ses composantes applique la traduction statistique au transfert depuis l'anglais vers CycL.

La traduction repose sur des modèles statistiques et utilise le système MOSES [Koehn *et al.*, 2007]. En vue d'entraîner ce système, de grandes quantités d'énoncés alignés (anglais $\leftrightarrow$ CycL) sont nécessaires. Les auteurs mettent alors en avant que la constitution manuelle d'une telle base d'énoncés alignés est trop coûteuse. L'alternative adoptée pour constituer un ensemble d'apprentissage de taille suffisante a été d'utiliser la génération de phrases en langue naturelle de l'ontologie Cyc.

L'exemple 1.5 donne un échantillon des sorties du système en CycL, étant données des assertions en anglais. L'évaluation automatique rapportée dans l'article [Tadić *et al.*, 2012] donne des scores plutôt bons compte tenu de la nouveauté de cette approche pour le transfert depuis la langue naturelle vers un langage formel. Le score BLEU atteint $65,26$ et le TER (taux d'erreurs/mots), $0,512$. Cependant, les méthodes d'évaluation automatiques tiennent peu compte de la cohérence syntaxique, et l'évaluation par annotateurs humains montre que la méthode de traduction statistique de MOSES cause beaucoup d'erreurs : "*almost 41% of all translations are CycL-nonfluent, thus breaching its syntactic rules (mostly due to the mismatching parenthesis)*" [5] [Tadić *et al.*, 2012].

1.3.3 Le transfert comme un problème de sélection

Par contraste avec la richesse de la langue naturelle en terme de quantité de formulations synonymes, le nombre de tâches distinctes dont un utilisateur peut avoir besoin est limité. L'ensemble des tâches est d'ailleurs très réduit dans certains cas comme lors de l'interaction avec des systèmes robotiques tels que ceux présentés en section 1.2 ainsi que d'autres dont il est discuté par la suite. Du fait

4. CycL est un langage formel déclaratif permettant de représenter les relations d'une ontologie. Il a été développé pour l'ontologie de connaissances communes Cyc (Cyc Language).

5. "presque 41% des traductions ne respectent pas le langage CycL (principalement à cause de problèmes de parenthésage)"

de la non ambiguïté des langages formels, un nombre de tâches réduit implique également un nombre réduit de commandes pour les exprimer[6]. Ce rapport entre une grande variété dans la langue source et un petit nombre de correspondances dans le langage cible implique que beaucoup d'énoncés en langue naturelle correspondent aux mêmes commandes. Cela rend pertinent de considérer le transfert comme un problème de recherche d'information [Manning *et al.*, 2008a], car le bruit entre les contextes (requêtes) proches n'est pas critique dans l'identification de la commande correcte.

Nous détaillons au chapitre 3 une approche fondée sur le principe de la recherche textuelle approchée parmi les requêtes pour identifier la commande à associer. Bien entendu, même si l'ensemble des tâches/commandes a une taille limitée, les possibilités qu'il offre à l'utilisateur sont bien plus étendues grâce à la variation des paramètres des requêtes. En posant le problème sous cet angle, le défi devient alors d'identifier correctement les paramètres dans la requête.

1.4 Interactivité

La capacité d'interaction naturelle est un atout important pour un assistant opérationnel. Elle passe comme nous l'avons vu par une expression de l'utilisateur à l'aide de la langue naturelle, mais peut être complétée par une expression du système lui-même dans la langue de l'utilisateur. L'objectif sous-jacent est d'ouvrir l'interaction sur un dialogue plus fluide, plus riche et intuitif pour l'utilisateur. La conception d'un système interactif et non seulement réactif permet en outre d'envisager la capacité du système d'apprendre incrémentalement. En effet, en situation interactive, il est possible d'obtenir immédiatement un retour de l'utilisateur sur la correction de la réaction du système. Le système peut alors mémoriser ce retour et utiliser cette information pour affiner ses prochaines propositions : dans le cas où le retour est négatif, il peut donner lieu à un apprentissage de la part du système. L'exemple 1.6 illustre ce type d'interaction par un échange avec un assistant opérationnel apprenant.

Dans la section suivante, nous présentons brièvement les alternatives de modélisation du dialogue de la littérature. Nous présentons dans ce qui suit un aperçu des développements apportés dans les champs traditionnellement opposés des systèmes de dialogue orienté-tâche et des systèmes de dialogue ouvert. Enfin, nous décrivons le modèle d'interaction retenu pour notre approche ainsi que les motivations de ce choix.

1.4.1 Modélisation du dialogue

Les systèmes d'interaction dialogique, orientés tâche ou non, peuvent généralement être regroupés en quatre groupes d'approches [Jokinen et McTear, 2009].

Les systèmes fondés sur des automates à états finis permettent un bon contrôle de l'interaction, la limitant de fait aux situations pré-établies dans les états de l'automate. Ils sont adaptés pour les tâches en domaine fermé mais le sont moins lorsque le dialogue nécessite plus de liberté. La taille des automates est un facteur fortement limitant. En effet, chaque nouvelle dimension de la situation dialogique cause globalement une multiplication du nombre d'états par le nombre de valeurs possibles. Ainsi si l'on considère le cas naïf d'un système capable d'accompagner interactivement l'utilisateur à travers les n étapes d'une réservation de billet, prendre en compte les m étapes de l'inscription de l'utilisateur au service peut requérir jusqu'à $(n+1) \times (m+1)$ états afin de considérer la combinaison de toutes les étapes intermédiaires possibles.

Les systèmes basés sur les structures de traits sont plus tolérants. Ils permettent aux étapes du dialogue de s'enchaîner dans un ordre arbitraire, mais toujours dans le cadre de dialogues en domaine fermé.

6. Selon les langages de programmation, le nombre de commandes peut être en réalité un peu plus élevé que le nombre de tâches réelles à cause de la définition de quelques alias (par exemple `dir` et `ls` en `bash`, ou les opérateurs `**` et `^` en `R`).

Utilisateur	Fais une copie du répertoire dev nommée dev_temp
Système	`cp -R dev dev_temp` `Est-ce correct ?`
Utilisateur	Oui
Système	`Que puis-je faire d'autre ?`
Utilisateur	Copie les fichiers csv depuis dev sur mon pc dans dev_temp
Système	`scp jarvis:dev/*.csv dev_temp/` `Est-ce correct ?`
Utilisateur	Non
Système	`Pouvez-vous m'enseigner la commande correcte ?`
Utilisateur	scp factoriel :dev/*.csv dev_temp/
Système	`Merci, je m'en souviendrai.` `Que puis-je faire d'autre ?`
Utilisateur	Supprime toutes les lignes de dev_temp/f.csv

Système	`Je ne suis pas certain, mais je pense à cette commande :`
	`echo -n > dev_temp/f.csv`
	`Est-ce correct ?`

Utilisateur	Oui
Système	`Que puis-je faire d'autre ?`
Utilisateur	Trie le contenu du fichier data.csv selon sa deuxième colonne

Système	`J'hésite entre plusieurs commandes :`
	`1- sort -k 2 -t ',' < data.csv`
	`2- sort -f 2 -d ',' < data.csv`
	`3- sort data.csv -t ',' -k 2`
	`Laquelle dois-je utiliser ?`

Utilisateur	La première.

EXEMPLE 1.6 – Interaction typique entre un utilisateur et un assistant opérationnel apprenant.

Les systèmes à base de plans s'appuient sur un ensemble de préconditions pour choisir à chaque acte de dialogue la ou les actions à effectuer. Les plans sont constitués d'une hiérarchie de dialogues et de sous-dialogues correspondant aux situations attendues.

Depuis une dizaine d'années, l'approche majoritaire pour les systèmes orientés tâches s'appuie sur les POMDP[7] [Young, 2006, Singh *et al.*, 2002, Schatzmann *et al.*, 2006]. Les POMDP modélisent les différents états et changements d'états possibles. La politique à suivre est déterminée grâce à un apprentissage par renforcement à l'aide de nombreuses expériences de dialogue avec des utilisateurs simulés.

1.4.2 Systèmes de dialogue à but de communication sociale

La famille des systèmes de dialogue en domaine ouvert regroupe les approches dont l'objectif est de maintenir une discussion aussi naturelle que possible. Certains travaux intègrent cet objectif dans le cadre d'un tâche spécifique [Devillers *et al.*, 2015], tandis que pour d'autres, aucun objectif opérationnel n'est défini au delà de la discussion sociale [Gilmartin et Campbell, 2016].

L'un des premiers systèmes de ce genre est ELIZA [Weizenbaum, 1966]. Il est construit sur un système de règles relativement simple et interagit en tenant le rôle d'un psychanalyste avec son interlocuteur qui prend donc le rôle de patient. L'exemple 1.7 présente un échantillon de discussion telle qu'elle peut être menée par ce système. Bien qu'il s'agisse d'un système en domaine ouvert car sup-

ELIZA :	Is something troubling you ?
Interlocuteur :	I may not worth what people expect of me.
ELIZA :	You say you may not worth what people expect of you ?
Interlocuteur :	Indeed.
ELIZA :	Do you feel strongly about discussing such things ?
Interlocuteur :	Actually, I'd rather not think about that.
ELIZA :	Can you elaborate on that ?
Interlocuteur :	What do you want to know ?
ELIZA :	Oh, I want to know ?
Interlocuteur :	Don't you ?
ELIZA :	You're not really talking about me – are you ?
Interlocuteur :	No, of course.
ELIZA :	You are being a bit negative.
Interlocuteur :	How can I improve this ?
ELIZA :	That's quite interesting.

EXEMPLE 1.7 – Interaction d'un "patient" avec le système ELIZA

posé réagir à tout type d'énoncé, son système de règles l'amène à rediriger l'interlocuteur vers des questions ou des sujets mieux maîtrisés par le système.

Récemment, des travaux ont réexaminé la question du dialogue en domaine ouvert en considérant des approches à base d'exemples. Plus précisément, il s'agit de retrouver parmi un corpus conséquent d'interactions dialogiques humaines l'énoncé le plus pertinent à proposer en réponse à l'interlocuteur [Lee *et al.*, 2009]. Les corpus de scripts de films ou de sous-titres de films et séries télévisées sont fréquemment utilisés, notamment avec les systèmes présentés dans [Banchs et Li, 2012, Ameixa *et al.*, 2014, Nio *et al.*, 2014]. Le chatbot UCAR[8] du projet JOKER exploite également une base de

7. Partially Observable Markov Decision Problems.
8. *Unsupervised Corpus-based Answer Retriever*

Interlocuteur :	hi there !
UCAR :	hi
Interlocuteur :	how are you Joker ?
UCAR :	i've got to get back to work
Interlocuteur :	yeah i've got to get back to work too
UCAR :	i'll go with you
Interlocuteur :	yeah sure

EXEMPLE 1.8 – Interaction en domaine ouvert avec UCAR

sous-titres en comparant différentes méthodes de sélection à partir de l'énoncé de l'interlocuteur [Dubuisson Duplessis *et al.*, 2016, Dubuisson Duplessis *et al.*, 2017]. Voir l'exemple 1.8 pour un aperçu du comportement dialogique du système.

Les approches à but de communication sociale permettent une interaction plus naturelle avec l'utilisateur, mais ne se focalisent pas sur l'accomplissement d'un objectif spécifique en dehors du dialogue lui-même. Elles ne conviennent donc pas *a priori* à notre problématique, qui implique de mener à bien les tâches d'exécution d'une commande et d'apprentissage interactif. Nous explorons dans la section suivante l'état de l'art à propos des systèmes de dialogue orientés tâche.

1.4.3 Systèmes orientés tâche

Les systèmes de dialogue orienté-tâche peuvent être considérés comme une extension des interfaces humain-machine classiques, grâce à l'utilisation de la langue naturelle. Un logiciel muni d'une interface conventionnelle attend en entrée l'ensemble des paramètres avec un format fixé. Pour un logiciel muni d'une interface dialogique, les paramètres sont extraits de l'interaction en langue naturelle avec l'utilisateur. Les questions posées par le système à l'utilisateur rendent intelligibles les paramètres requis, et dispensent (souvent) de les formater de manière spécifique. D'autre part, les systèmes de dialogue orienté-tâche peuvent être capables d'expliquer en langue naturelle également les opérations qu'ils effectuent, voire les raisons qui les y poussent. Ces capacités rendent l'usage de ces systèmes accessible à des utilisateurs non-experts.

Le système SHRDLU [9] développé par Terry Winograd [Winograd, 1972] a pour fonction de répondre à des questions et exécuter des commandes par le biais d'un dialogue en anglais, voir un échantillon dans l'exemple 1.9. Les questions et les commandes doivent porter sur la représentation du monde des blocs maintenue par le système, illustrée sur la figure 1.2.

Winograd fonde son approche sur une analyse de l'anglais à l'aide d'une grammaire, mais souligne l'importance de la généricité pour la conception de systèmes de compréhension de la langue, les raccourcis *ad hoc* doivent être évités. Cette approche ne permet pas l'adaptation à de nouvelles capacités communicatives (par exemple une nouvelle langue) ou à une nouvelle tâche (par exemple l'opération "clone" ou encore une gestion temporelle des blocs).

Un autre système interactif historique a été décrit peu après dans [Shortliffe *et al.*, 1975]. Il s'agit de MYCIN, un système de diagnostic médical et de prescription d'antibiotiques. MYCIN est un système expert, c'est-à-dire un système de raisonnement sur une base de connaissances structurée. Il pose une série de questions à l'utilisateur en vue de déterminer par inférences probabilistes l'hypothèse la plus plausible (ainsi que des alternatives) concernant la cause des symptômes, et d'en déduire le traitement

9. Il ne s'agit pas d'un acronyme, le nom provient en réalité de l'agencement des lettres sur les claviers de Linotypes anglophones.

Utilisateur :	*pick up a big red block*
SHRDLU :	`ok`
Utilisateur :	*grasp the pyramid*
SHRDLU :	`I don't understand which pyramid you mean`
Utilisateur :	*find a block which is taller than the one you are holding and put it into the box*
SHRDLU :	`ok`
Utilisateur :	*what does the box contain ?*
SHRDLU :	`the blue pyramid and the blue block`
Utilisateur :	*what is the pyramid supported by*
SHRDLU :	`the box`
Utilisateur :	*how many blocks are not in the box*
SHRDLU :	`four of them`

EXEMPLE 1.9 – Dialogue avec SHRDLU

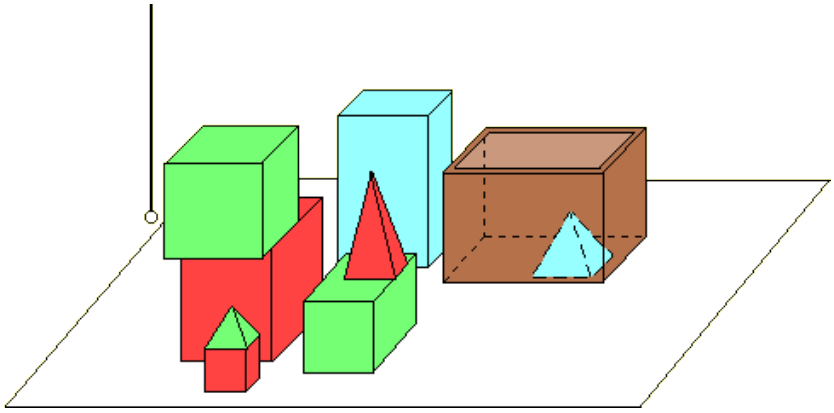


FIGURE 1.2 – Dialogue avec SHRDLU et représentation graphique associée du monde des blocs

adapté. Il fournit également un indice de confiance associé à chacun de ses diagnostics. Ce système d'aide au diagnostic, malgré son efficacité (il a été jugé meilleur qu'un ensemble de spécialistes du domaine [Yu *et al.*, 1984]), n'a pas la capacité d'apprendre par son interaction. Toute modification de ses connaissances en vue d'améliorer ses performances nécessite l'intervention d'un expert.

L'interaction dialogique naturelle a également été appliquée pour la tâche de réponse à des questions en domaine ouvert [Rosset *et al.*, 2006a, van Schooten *et al.*, 2007]. Le projet RITEL (recherche d'informations par téléphone) intègre la capacité de répondre aux questions de l'utilisateur et celle d'interagir à l'oral en langue naturelle. Il combine donc les défis de chacun de ces domaines : détection de parole, reconnaissance de la parole, analyse linguistique, recherche d'informations pertinentes, génération linguistique et enfin synthèse vocale.

Chacune de ces tâches est traitée par un module distinct utilisant une méthode spécifique. Certains d'entre eux sont fondés sur des modèles statistiques (reconnaissance audio, recherche d'informations), d'autres sur des bases de règles (analyse linguistique, gestion du dialogue). La pertinence des réponses apportées par le système dépend souvent de la combinaison de l'ensemble de ces modules. Néanmoins, considérons la capacité d'apprentissage par exemple pour le module de recherche d'informations seulement, car il constitue le "cœur de métier" du système. Rendre ce module capable d'apprendre par l'interaction requiert de disposer d'une évaluation pour les réponses, aussi peu influencée que possible par les autres modules. Alors, le système de recherche d'informations peut ajuster les

- What <businesses> are within <distance> of <address> ?
- What is the most expensive purchase approved between <start date> and <end date> ?
- Find articles related to <topic> written for <project>.
- Which <project> had the greatest travel expenses between <start date> and <end date> ?

EXEMPLE 1.10 – Échantillon de requêtes utilisées pour l'évaluation du système PLOW

poids de certains termes en fonction du retour obtenu.

Il s'agit d'une méthode d'apprentissage naïve et on peut s'attendre à ce que son efficacité soit limitée. Cela nous permet néanmoins d'illustrer la difficulté d'introduire un mécanisme d'apprentissage pour des systèmes complexes tels que RITEL.

Plus récemment a été proposée une approche pour un assistant opérationnel (appelé par les auteurs *Procedural Dialog System*) à visée plus générique [Volkova *et al.*, 2013]. Elle est fondée sur l'extraction d'arbres procéduraux à partir de documents d'aide ou de spécification. Un arbre procédural représente les choix (nœuds) et les actions (liens) qui peuvent être envisagés dans une activité donnée. La structure dialogique est ensuite calquée sur ces arbres. On cherche le plus profond des nœuds d'un arbre correspondant à la requête de l'utilisateur. Le dialogue est ensuite dirigé à partir de la structure du sous-arbre de ce nœud.

Les travaux que nous venons de présenter permettent d'effectuer une tâche en suivant les instructions de l'utilisateur. Cependant, ces systèmes ne peuvent en apprendre de nouvelles.

1.5 Apprentissage interactif

Nous proposons de définir un **assistant opérationnel incrémental**[*] comme un assistant opérationnel capable d'apprendre par interaction avec son utilisateur. Nous pouvons alors reformuler l'objectif de ce travail comme la conception d'un assistant opérationnel incrémental générique, et bien entendu aussi efficace que possible. La généricité et l'efficacité sont généralement inversement corrélées, et nous proposons de focaliser notre objectif sur l'aspect de généricité.

1.5.1 PLOW : Un exemple d'assistant opérationnel incrémental

Allen et collègues ont proposé [Allen *et al.*, 2007] un assistant opérationnel pour la navigation Internet (appelé PLOW) capable d'apprendre de nouvelles tâches par l'interaction avec son utilisateur. Les tâches considérées consistent en la recherche ou l'extraction d'informations dans les pages ou sites web à partir de requêtes formulées en langue naturelle, voir l'exemple 1.10 pour un aperçu du type de requêtes considéré.

Lorsqu'une requête ne peut pas être résolue, le système démarre avec l'utilisateur une phase d'apprentissage dans laquelle l'utilisateur doit enseigner à PLOW la séquence d'actions nécessaire pour obtenir le résultat souhaité. L'utilisateur interagit pour cela de manière multimodale avec le système. Ce dernier peut ainsi tirer avantage à la fois des instructions données par l'utilisateur en langue naturelle dans l'espace de saisie, et des actions qu'il effectue dans la fenêtre de navigation et qui sont enregistrées simultanément. Lors de cette phase d'apprentissage, le problème est résolu de manière collaborative avec l'utilisateur, le système demandant des explications quant aux opérations effectuées. PLOW peut en outre s'appuyer sur des opérations déjà apprises, y compris dans la phase d'apprentissage courante, pour rendre le processus d'apprentissage plus rapide et moins fastidieux.

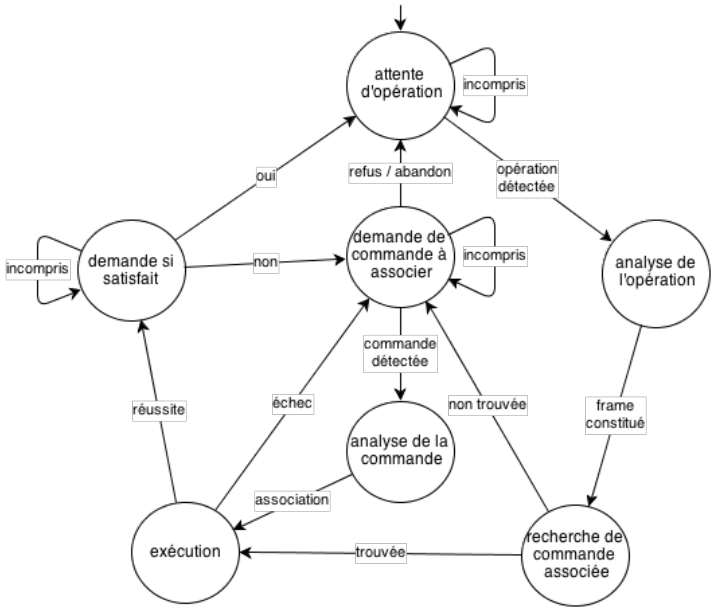


FIGURE 1.3 – Automate du protocole général d'interaction

Le système PLOW est rendu spécifique à son domaine par la multimodalité qu'il permet d'utiliser pour les phases d'apprentissage. Afin de respecter la contrainte de généricité fixée en introduction, nous avons développé un modèle d'apprentissage interactif simple, décrit dans la section suivante.

1.5.2 Problématique : vers une première modélisation

La question de l'apprentissage artificiel par le dialogue implique de définir un protocole interactif adapté. La séparation des phases d'apprentissage et d'utilisation rend l'interaction fastidieuse au cours de l'étape d'amorçage des connaissances. Il est notamment courant de définir pour l'amorçage un programme d'apprentissage [10], qui doit être suffisamment précis et complet pour couvrir une proportion raisonnable de cas au cours de l'utilisation. L'interaction au cours de l'étape d'amorçage est en outre limitée en spontanéité et en initiative de la part de l'utilisateur ; il est alors discutable que cette étape relève réellement du dialogue. D'autre part, un apprentissage uniquement au cours de l'étape d'amorçage est souvent inefficace à long terme. C'est le cas en particulier sur un domaine ouvert car il est trop vaste (connaissances générales, langue naturelle) ou évolutif (nouvelles technologies, actualités).

C'est l'objectif d'éviter cette étape d'amorçage et d'assouplir l'interaction entre le système et l'utilisateur qui motive l'emploi de l'apprentissage incrémental. Seul un modèle incrémental mixte, c'est-à-dire mêlant utilisation et apprentissage, permet de maintenir un ancrage continu des connaissances du système aux besoins de ses utilisateurs.

Ce modèle incrémental est décrit par l'automate de la figure 1.3. Il consiste en une boucle principale de requêtes à l'initiative de l'utilisateur. Pour chaque requête soumise, le système cherche un ensemble de réponses à proposer. L'utilisateur désigne ensuite une réponse de son choix, ou la valide si une seule réponse est proposée. Enfin, une opération de vérification peut être ajoutée après le choix afin d'éviter qu'une erreur de la part de l'utilisateur [11] ne conduise à l'ajout d'une erreur à la

10. On appelle ici *programme d'apprentissage* la liste des tâches qui sont enseignées au système.

11. Le type d'erreur désigné ici correspond aux fautes de frappe/de clic, ou à une décision prise trop rapidement au sujet de la commande correcte.

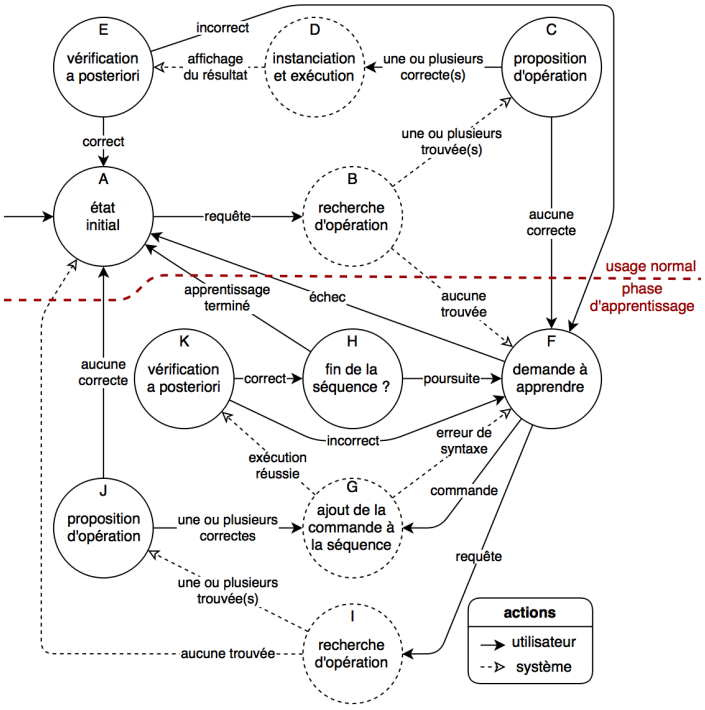


FIGURE 1.4 – Automate du protocole d'interaction avec apprentissage récursif de séquences

base de connaissances. À l'issue de cette boucle, l'association entre la requête de l'utilisateur et la réponse désignée est ajoutée à la base d'exemples. Le système peut échouer à proposer une commande à l'utilisateur, et donc à l'ajouter à sa base de connaissances, dans trois cas de figure :

- l'ensemble de réponses proposées par le système est vide ;
- aucune des réponses proposées par le système n'a été désignée comme valide par l'utilisateur ;
- la réponse désignée comme valide s'est révélée incorrecte *a posteriori* [12].

Ces situations conduisent à la seconde boucle du système : la boucle d'apprentissage. L'utilisateur est alors sollicité afin de proposer une solution à la requête que le système vient d'échouer à résoudre. Alors que le modèle présenté sur la figure 1.3 ne présente qu'une seule étape pour l'ajout d'une nouvelle association, une extension de celui-ci sur la figure 1.4 permet d'ajouter une séquence et d'utiliser éventuellement la langue naturelle. Dans ce second modèle, deux options sont possibles pour effectuer cette complétion : demander directement à l'utilisateur la réponse attendue, ou demander une reformulation de la requête qui permettrait de chercher à nouveau une réponse à l'aide de la base d'exemples. Dans le premier cas, l'association est simplement ajoutée à la base. Dans le second cas, le système répète une séquence d'opérations similaire à la boucle principale à la différence qu'en cas de succès, deux exemples sont ajoutés à la base. Ils correspondent aux deux reformulations de la même requête, associées avec la même réponse.

Nous avons présenté un modèle incrémental fondé sur une interaction mêlant exploration et exploitation, c'est-à-dire apprentissage et utilisation du système. Nous nous appuierons sur ce modèle pour étudier l'apprentissage du système au contact des utilisateurs.

12. Il s'agit du cas dans lequel l'utilisateur non expert aura sélectionné (ou enseigné) une commande dont l'exécution par le système ne produit en réalité pas le résultat attendu. Il s'agit donc d'une erreur de l'utilisateur, mais qu'il convient bien entendu de prendre en compte également.

Conclusion

Nous avons examiné dans ce chapitre l'état de l'art dans le domaine de la programmation automatique, constatant que la programmation en langue naturelle découle directement de ce domaine. Considérant la question de l'apprentissage par interaction dans le cadre de ce domaine, nous nous sommes intéressé aux méthodes permettant de produire une instruction en langage formel à partir d'une requête en langue naturelle.

En particulier, parmi celles-ci, nous retenons une approche par sélection, issue du domaine de la recherche d'information, ainsi que les approches de traduction à base de cas indépendantes du langage par analogie. L'apprentissage par interaction implique des contraintes sur l'interaction du système avec l'utilisateur. Après avoir présenté les stratégies existantes de gestion de l'interaction, et notamment du dialogue en langue naturelle, nous adoptons une stratégie de gestion dirigée par le système. En effet, il nous semble que l'apprentissage de la tâche de transfert requiert un cadre interactif contrôlé.

L'évaluation des performances du système de transfert depuis langue naturelle vers langage formel passe par la comparaison des résultats du système avec une référence [13]. Tester l'apprentissage requiert d'autre part un ensemble d'exemples pour constituer la base de connaissances du système. Ces ressources sont rares pour cette tâche en particulier. Le chapitre suivant est dédié à la collecte de bases d'exemples.

13. Le terme référence désigne ici l'ensemble des réponses attendues.

Chapitre 2

Collecte de bases d'exemples

Sommaire

2.1 Introduction

Un système capable de transférer un langage vers un autre indépendamment des langages source et cible ne dispose pas de connaissances *a priori*. Il doit donc les extraire d'une base d'exemples externe pour y adapter son comportement. En outre, l'évaluation du système nécessite également un ensemble d'exemples représentant les capacités attendues. Il est donc nécessaire de disposer d'une base d'exemples adaptée à la tâche qui nous intéresse : le transfert de requêtes en langage naturel vers des commandes en langage formel. Le comportement d'un système apprenant dépend de deux facteurs d'importance équivalente : d'une part le type d'apprentissage artificiel choisi conditionne la manière dont les connaissances sont exploitées, et d'autre part la constitution de la base d'exemples elle-même joue un rôle central dans la capacité à produire des réponses. Nous traiterons du choix de la méthode d'apprentissage dans le chapitre suivant ainsi que dans la deuxième partie. Dans le présent chapitre, nous nous intéressons au deuxième facteur : la constitution de la base d'exemples. Nous proposons un format et une méthode de collecte pour permettre une utilisation générique et minimiser les biais influençant le contenu de la base.

L'objectif général considéré est de doter un système, par apprentissage, de réactions appropriées à des sollicitations d'utilisateurs sous forme de requêtes en langage naturel. La manière de formuler les requêtes peut varier d'une personne à l'autre et d'une tâche (commande) à l'autre. Le **naturel des exemples**, ou leur adéquation avec la variété des formulations par les utilisateurs, est donc une propriété déterminante pour la qualité de la base. Dans l'optique de limiter les biais expérimentaux, la **minimisation du bruit** est également à prendre en compte, c'est-à-dire la minimisation du nombre d'exemples erronés dans la base, à moins qu'ils ne soient étiquetés comme tels. D'autre part, la **quantité d'exemples disponibles** est bien sûr un critère à maximiser. Enfin, pour des raisons pratiques de mise en place expérimentale et, par la suite, de reproductibilité, l'**efficacité de la collecte** devrait être maximisée également. On entend par efficacité la vitesse à laquelle les exemples peuvent être rassemblés et la spécificité du traitement que doivent subir les données avant d'être utilisables. Autrement dit, on préférera *a priori* l'assemblage à partir de données existantes à une collecte plus lente directement auprès de volontaires. De même, on préférera plutôt une source de données déjà formatées en associations de requêtes avec des commandes à une source nécessitant d'identifier les informations pertinentes pour ensuite les extraire et les formater correctement. Ce dernier cas nécessite en effet la mise au point d'un algorithme efficace, apprenant ou à base de règles, pour gérer ces tâches complexes. Cet aspect est développé et illustré en section 2.3.

Nous proposons dans la suite un tour d'horizon des ressources existantes et des possibilités de collecte en examinant pour chaque cas ses qualités pour les quatre propriétés que nous avons citées.

2.2 Format

Évaluer la qualité de sources potentielles ne peut se faire qu'en considérant le format final choisi pour la base d'exemples. Cette question est déterminante pour le processus d'acquisition des exemples. En effet, un changement dans le format des exemples peut nécessiter l'organisation d'une toute nouvelle collecte. C'est le cas en particulier lorsqu'une nouvelle information contextuelle (par exemple un identifiant de l'utilisateur, un historique d'utilisation, ou encore une localisation géographique) est prise en compte dans le traitement de chaque exemple [1].

1. Remarquons toutefois qu'il est possible dans certains cas de rendre le système tolérant à la présence ou l'absence d'une information contextuelle particulière, par exemple en l'ignorant ou en prenant une valeur par défaut en cas d'absence. L'inconvénient de cette dernière option est que le système est biaisé par la proportion des exemples de la base contenant cette information supplémentaire, tandis que la première option ignore totalement l'ajout de cette information.

Dans notre cas, chaque exemple doit contenir la requête de l'utilisateur et une commande correcte qui lui correspond. Les commandes seront représentées par leur chaîne de caractères. Comme annoncé en introduction, nous considérons pour l'expression de l'utilisateur la modalité écrite. Les langages de programmation sont des langages écrits, qui emploient notamment des caractères non alphanumériques et une syntaxe spécifique. La formulation des requêtes en langue naturelle suit cette influence, au moins à cause des paramètres des commandes qu'elles contiennent fréquemment. Ces paramètres peuvent être des nombres ou des noms (de fichiers, répertoires, URLs, ...) pouvant ne correspondre à aucun mot de la langue naturelle employée, voir d'aucune langue existante, et contenir des caractères non alphanumériques. La modalité orale n'est donc *a priori* pas idéale pour l'expression des requêtes, car les paramètres ne sont en général pas prononçables simplement comme des mots. En outre, l'oral pour l'expression de requêtes opérationnelles pourrait manquer de fonctionnalités utiles telles que la navigation dans les requêtes précédemment énoncées ou l'édition de la requête courante avant de la soumettre.

La commande vocale d'un système assistant reste une perspective passionnante, mais elle nécessite, pour rester générique, une chaîne de traitements non supervisée efficace. Avant de s'engager sur cette voie, il convient dans un premier temps de concevoir une approche satisfaisante sur le problème de transfert depuis la langue naturelle écrite vers le langage formel. D'ici là, nous choisissons d'écarter de notre étude la modalité orale, et avec elle la représentation phonétique des requêtes en langue naturelle, qui seront donc représentées par des chaîne des caractères.

Par ailleurs, étant donnée la nature dialogique de la tâche, il est possible d'intégrer le contexte du dialogue dans le format des exemples. Le contexte peut contenir non seulement l'historique des requêtes soumises et commandes exécutées, mais aussi des informations supplémentaires concernant l'utilisateur et l'environnement d'exécution. Cependant, la multiplication des informations disponibles dans chaque exemple rend plus difficile la détection des traits pertinents par une méthode d'apprentissage. Les problèmes éthiques s'ajoutent au choix de ne pas enregistrer les informations à propos des utilisateurs. De plus, l'interprétation des requêtes hors contexte, bien que limitée, nous semble être un champ déjà vaste à explorer.

Nous avons également choisi de ne pas inclure au format des exemples d'information précalculée pour indexer, réduire ou analyser le contenu de la base. Les traitements permettant d'obtenir ces informations supplémentaires sont susceptibles d'être édités, ajoutés ou retirés, modifiant en conséquence le format à prendre en compte pour l'opération de transfert. De plus, certains de ces traitements sont dépendants de la langue utilisée pour les requêtes. Les seuls éléments toujours présents et inaltérés de la base d'exemples sont donc les chaînes de caractères des requêtes et celles des commandes.

La simplicité voulue dans le format des exemples, et par suite dans les approches que nous proposons, est motivée par le souci de contrôler autant de biais que possible dans les expériences. Les domaines liés au langage naturel ont en effet pour caractéristique d'apporter déjà bon nombre de biais difficilement contrôlables comme nous en discutons en section 2.5.

2.3 Sources

Les bases d'associations entre requêtes en langue naturelle et commandes en langage formel sont plutôt rares car il s'agit d'un domaine encore peu abordé, sinon dans des cadres plus spécifiques, *cf.* section 1.3.2. Deux pistes peuvent être envisagées pour constituer une base d'exemples pour un système de transfert. La première consiste à assembler les exemples à partir de ressources préexistantes telles que le *Linux Plan Corpus* [Blaylock et Allen, 2004], les forums d'aide en ligne comme le site `www.stackoverflow.com`, ou les historiques de commandes accessibles par exemple sur les dépôts publics de `github.com`. Cette première piste entraîne une qualité variable des données ou de

leur format à cause des adaptations nécessaires et de la quantité de bruit présent.

La seconde piste consiste à collecter directement les exemples auprès d'utilisateurs volontaires. C'est celle-ci que nous détaillons dans la suite.

2.4 Méthodes de collecte

Nous avons collecté des associations entre commandes en langage formel et requêtes en langue naturelle auprès de volontaires. Trois corpus ont été constitués, avec des protocoles de collecte différents, induisant donc des biais différents pour chacun. Le tableau 2.2 résume leurs caractéristiques principales, le nombre d'énoncés source uniques a été déterminé par comparaison exacte des chaînes de caractères, tandis que le nombre d'énoncés cible uniques a été obtenu en ignorant les paramètres des commandes. Par exemple, les commandes "`cat -n fichier.txt`" et "`cat < data.log`" sont considérées identiques, alors que "`cat data.tsv | cut -f 3`" est différente. Les sections qui suivent décrivent ces trois corpus avec plus de détails [2].

2.4.1 Corpus R : équilibré

Le premier corpus collecté associe des commandes dans le langage `R` [3] à des requêtes en français. Le langage `R` est très complet, permettant un grand nombre de traitements sur des données de types et de dimensions variés, ainsi que leur représentation graphique. La motivation du choix de ce langage vient de son utilisation plutôt occasionnelle [4], rendant plus difficile la mémorisation de ses fonctions complexes par les utilisateurs. Il s'agit néanmoins d'un langage plutôt largement utilisé pour les traitements statistiques.

Le corpus est composé de 525 associations uniques pour des commandes toutes différentes, bien que quelques unes soient synonymes. Elles ont été collectées par un seul participant. L'objectif de cette collecte était de maximiser la couverture des commandes existantes grâce au protocole suivant : nous avons extrait d'un site web de tutoriels sur le langage `R` un ensemble de commandes avec leurs descriptions. Les descriptions ont ensuite été reformulées lorsque c'était nécessaire : lorsque la description initiale n'était pas suffisamment claire ou complète, elle était modifiée pour respecter une formulation autosuffisante de type impérative/infinitive. L'exemple 2.1 présente un échantillon de commandes et leurs descriptions extraites du site, ainsi que leurs versions reformulées intégrées au corpus.

2.4.2 Corpus bash1 : reformulations

Le deuxième corpus a été collecté avec le nouvel objectif d'introduire de la variation dans les requêtes en langue naturelle. Les requêtes sont toujours en français, mais les commandes sont en `bash`. Ce changement de langage formel est motivé par la taille de la communauté des utilisateurs de `bash`, ainsi que l'expressivité, la modularité et la relative simplicité du langage. Ces conditions rendent plus aisée la collecte auprès de volontaires maîtrisant ce langage de script. Les participants ont reçu des exemples d'associations requête-commande avec la consigne de rédiger des reformulations des requêtes leur paraissant naturelles. Aucune autre directive n'a été donnée sur la rédaction des reformulations. À l'issue de la collecte effectuée par 3 participants, le corpus comprenait 512 associations dont 486 uniques ; l'exemple 2.2 en présente un échantillon. Les volontaires ont en effet parfois proposé les mêmes reformulations, ce qui s'explique par le fait qu'ils s'appuyaient tous initialement sur les mêmes exemples de requête. D'autre part, le nombre plutôt élevé de reformulations par requête rend

2. Les corpus collectés sont accessible sous licence libre à l'adresse `http://anelda.limsi.fr`
3. `www.r-project.org`
4. Cette tendance observée ne vaut pas généralisation.

	requête	commande
initiale	vecteur dont chaque coordonnée est l'inverse	`1 / x`
reformulation	inverse X	`1 / X`
initiale	donne un vecteur avec les 5 premières valeurs de x	`x[1:5]`
reformulation	donne un vecteur avec les 5 premières valeurs de X	`X [ 1 : 5 ]`
initiale	donne (1.0, 1.5, 2.0)	`seq(from = 1, to = 2, by = 0.5)` ou `seq(1, 2, 0.5)`
reformulation	donne le vecteur de 2 à 12 par pas de 5	`seq ( 2 , 12 , 5 )`

EXEMPLE 2.1 – Associations extraites pour le corpus `R`

1 2 3	liste le contenu du répertoire courant liste les fichiers liste les fichiers s'il-te-plaît	`ls`
4 5	renomme le fichier foo en bar peux-tu renommer foo en bar	`mv foo bar`
6	quelles lignes contiennent foo et bar dans foo	`cat foo | grep "foo" | grep "bar"`
7 8	va dans le directory idial et récupère la dernière version va dans le directory idial et fais un pull	`cd idial; git pull`

EXEMPLE 2.2 – Associations dans le corpus `bash1`. Numérotation des lignes arbitraire. Échantillon non représentatif de la distribution du nombre de reformulations par requêtes.

également les collisions plus probables. Le corpus comptait en `bash` 20 commandes distinctes, chacune associée avec entre 1 et 79 des requêtes reformulées (avec une médiane de 27). Nous donnons ci-dessous la liste de ces commandes :

```
cat foo
cat foo | grep "foo" | grep "bar"
cd idial; git pull
cd nelda; find . -name "*nelda_v1.5*"
cp foo bar
echo "# Fonctionne uniquement en cas d'échec." >> foo.py
file foo
find . -iname "*foo*"! -iname "*bar*" -exec mv {} .. \;
find nelda -name "*nelda_v1.5*"
if [ "$FOO" = "" ]; then export FOO=42; fi
ls
mv foo ..
mv foo bar
rm *.txt
rm a.txt b.txt
rm a.txt b.txt c.txt
rm foo
trash-put foo
wc -w foo
wget perso.limsi.fr/user/biblio.zip; unzip biblio.zip
```

Aucune modification n'est opérée sur les commandes selon la requête qui lui est associée. Nous avons tenté de composer cette liste de commandes afin de couvrir les opérations les plus essentielles d'un système d'exploitation, pour qu'elles soient ensuite déclinées avec des paramètres différents. Cependant, il s'agit toujours d'une liste arbitraire dont nous n'avons pas mesuré si elle correspond à l'usage réel que font les utilisateurs des commandes `bash`.

2.4.3 Corpus bash2 : indépendance des participants

Le dernier corpus collecté associe également des requêtes en français à des commandes en `bash`. Il contient 578 associations uniques rédigées par 7 personnes. L'objectif est ici d'éviter le biais influençant les formulations des participants ainsi que le choix et la distribution des commandes.

En effet, dans le corpus `bash1`, les participants reprenaient souvent les requêtes déjà présentes, qu'ils pouvaient consulter, auxquelles ils appliquaient des modifications mineures [5], telles que mises en évidence dans l'exemple 2.2. Le corpus `bash2` a été collecté en tentant d'éliminer ce biais, pour laisser le plus de liberté possible à la formulation d'une requête par les participants. La collecte a donc été organisée de manière à ne donner aux participants aucune influence quant à la forme des requêtes en langue naturelle qui étaient attendues. Seules des commandes étaient présentées aux participants, avec pour consigne de formuler pour chacune une requête en langue naturelle en s'imaginant s'adresser à une autre personne. Aucun accès n'était donné aux associations déjà collectées, y compris par le même participant. Seuls trois exemples minimaux étaient joints à la consigne, le guide de contribution complet est joint en annexe B. Bien entendu, la mémoire des requêtes déjà données est susceptible d'influencer les suivantes. Ce biais ne peut pas être évité, mais on peut le considérer comme non problématique car les premières requêtes soumises par le participant et qu'il a en mémoire ne sont, elles, soumises à aucune influence spécifique. Les formulations des participants pour les requêtes sont donc plus naturelles que lors des collectes précédentes. Il est notamment plus fréquent d'identifier des syntaxes nouvelles ou atypiques, mais cela n'est pas un inconvénient car nous adoptons la démarche

5. Ces modifications suivaient d'ailleurs très fréquemment les principes de l'analogie formelle, voir chapitre 4.

descriptive [6] de la langue pour une meilleure adéquation avec les usages.

1	Va dans le dossier test/.	`cd test/`
2	va dans le dossier git	`cd git/`
3	vers analogy/	`cd analogy`
4	affiche l'état de l'index git	
5	affiche le statut du dépôt git	`git status`
6	vérifie l'état des différents fichiers dans le git	
7	compte les mots dans RAD.tex et dialog.tex	`wc RAD.tex dialog.tex`
8	j'veux regénérer les locales	`locale-gen`
9	teste si anonymized-domain.org est accessible	`ping anonymized-domain.org`
10	fais un ping sur www.google.fr	`ping www.google.fr`

EXEMPLE 2.3 – Échantillon d'associations obtenues lors de la collecte BASH2. Numérotation des lignes arbitraire. Échantillon non représentatif de la distribution du nombre de reformulations par requêtes.

Par ailleurs, contrairement à la collecte `bash1` pour laquelle la liste des commandes considérées était fixe et imposée aux participants, la collecte `bash2` permet plus d'ouverture et limite donc ce biais. En effet, les commandes présentées aux participants pour formuler les requêtes sont une sélection aléatoire directement extraite de leur fichier `.bash_history` [7], avec anonymisation si nécessaire. Ainsi, l'éventail des commandes mises en jeu est non seulement plus grand, avec 98 têtes de commandes uniques pour `bash2` contre 20 dans `bash1`, mais également plus représentatif des usages au point de vue de la distribution. Les plus hautes fréquences de l'histogramme des commandes sont données dans le tableau 2.1. Seules les têtes des commandes sont utilisées pour établir l'histogramme, les commandes complètes se répétant rarement du fait de leurs paramètres variés.

2.5 Analyse des bases collectées

Dans cette section, nous discutons des propriétés qui rendent un corpus représentatif des phénomènes qu'il est censé couvrir à partir des travaux abordant cette question dans l'optique de l'étude du langage naturel. Nous constatons également que les outils les plus avancés de mesure de la représentativité et de constitution des corpus de langue naturelle ne peuvent pas être appliqués au cas d'un corpus d'associations entre requêtes et commandes.

Nous présentons donc les corpus de référence que nous avons constitués pour ce cadre, et proposons une analyse de la représentativité de nos propres corpus d'associations par rapport à ces derniers à l'aide d'une mesure la plus neutre possible : l'étude comparative de la forme des histogrammes de leurs traits.

2.5.1 Le problème de la représentativité

De nombreux systèmes de traitement du langage naturel, notamment en traduction automatique, analyse ou réponse à des questions, s'appuient sur l'utilisation de corpus collectés pour modéliser leur tâche. La représentativité des corpus par rapport à la distribution réelle des usages est un facteur important pour l'acquisition d'un modèle adapté, et donc efficace.

6. Démarche descriptive vs. démarche prescriptive : voir 1.3.1.
7. Il s'agit d'un fichier contenant la liste des dernières commandes utilisées sous Linux.

occurrences	tête de la commande
102	`cd`
55	`git`
51	`ls`
35	`cat`
27	`less`
25	`ssh`
17	`vim`
14	`ipython`
13	`mv`
10	`perl`
10	`grep`
10	`emacs`
8	`wc`
8	`man`
8	`cp`
7	`echo`

TABLEAU 2.1 – Liste des 16 commandes les plus fréquentes dans le corpus `bash2`

		R	bash1	bash2
participants		1	3	7
associations		525	512	578
requêtes (uniques)		495	486	541
commandes (uniques)		204	20	98
total de mots (requêtes)		4029	5071	3799
longueur	moyenne	7,7	9,9	6,6
	médiane	7	9,5	5
	écart-type	3,2	4,5	4,5

TABLEAU 2.2 – Corpus collectés

Pour la tâche définie, nous avons besoin d'un corpus aligné entre langage naturel et langage formel. À notre connaissance, il n'existe pas encore d'étude portant sur la représentativité de tels corpus.

Les travaux de Douglas Biber [Biber, 1993] abordent le problème de la représentativité sous l'angle de la constitution de corpus de langue naturelle pour l'étude, notamment statistique, des phénomènes linguistiques. Il définit la **représentativité*** comme le degré auquel un échantillon inclut l'étendue complète de la variabilité au sein d'une population. La condition généralement admise pour qu'une méthode d'échantillonnage soit considérée comme représentative est qu'elle soit proportionnelle. Lors de la collecte de corpus de langue naturelle, l'échantillonnage démographique, *i.e.* un tirage aléatoire indépendant dans un ensemble de grande taille de la population visée, est représentatif par définition. Biber mentionne le problème de l'influence inégale des différentes sources de textes, qui requiert une autre notion de représentativité pour les corpus de langue naturelle. Dans notre cadre cependant, l'existence de différents types de paires requête-commande qui pourraient biaiser l'équilibre des phénomènes dans le corpus n'est pas évidente.

Il existe également plusieurs méthodes pour déterminer la taille optimale de l'échantillon, sur lesquelles nous ne nous étendrons pas. Il apparaît en effet évident que les données dont nous disposons actuellement sont sous-échantillonnées.

D'autres travaux abordent la question de la représentativité des corpus, notamment [Jones et Galliers, 1996, Leech, 2006], mais toujours en se focalisant sur la linguistique de corpus. La plupart des aspects discutés sont donc difficilement applicables pour mesurer la représentativité d'un corpus d'associations entre langue naturelle et langage formel, d'autant que la langue naturelle des requêtes est implicitement mais fortement contrainte. Une étude plus récente [Rapp, 2014] propose une mesure de représentativité fondée sur la notion de familiarité des mots pour les lecteurs et sur leurs associations lexicales spontanées. Ici aussi, de même que nous l'avons remarqué, l'application à une tâche spécifique pose question, notamment car les associations spontanées font intervenir des connaissances générales qui sortent donc du champ couvert par un corpus d'associations requêtes-commandes.

Aucun corpus ne peut être parfaitement représentatif au sens strict [Cappeau et Gadet, 2007], mais il est possible de comparer les corpus entre eux pour tenter d'évaluer leurs représentativités relatives l'un par rapport à l'autre, à l'aide de la proximité de leurs traits. Pour en tirer une conclusion, cela nécessite bien entendu que l'un des deux soit un corpus de référence, largement échantillonné et dont on admettra qu'il est suffisamment représentatif.

Dans le cadre du transfert de langages, deux aspects doivent être considérés : la représentativité relative spécifique du corpus pour la langue source, et pour la langue cible. Cela s'applique également dans notre cas sur la langue naturelle d'une part et le langage formel d'autre part, à une nuance près : l'étendue des énoncés susceptibles d'être transférés de la langue naturelle au langage formel est très restreinte par rapport à l'ensemble des énoncés observables dans la langue en général, et il faut en tenir compte lors de la mesure de représentativité relativement à un corpus de référence général.

Mesurer une représentativité relative entre deux corpus peut bénéficier des travaux sur l'identification d'auteurs tels que ceux de [Juola et Baayen, 2005]. Cette méthode pose problème pour les domaines disposant de peu de données, car le modèle requiert une quantité importante de texte brut ainsi que si possible des énoncés longs et enchaînés pour une comparaison efficace. Elle n'est donc pas adaptée pour notre corpus de requêtes, ni *a fortiori* pour les commandes.

La distribution des n-grammes dans les corpus est considérée comme un trait caractéristique pertinent et fréquemment utilisé pour l'analyse comparative de corpus. Son avantage est qu'elle est adaptable au langage formel et ne requiert pas *a priori* d'énoncés d'une longueur définie. Nous avons choisi de comparer la forme des histogrammes de traits entre les demi-corpus collectés et deux corpus de référence sélectionnés. Nous comparons également les occurrences obtenues avec celles prédites par la loi de Zipf [Zipf, 1935], aussi appelée loi d'Estoup-Zipf, ou loi de Pareto-Yule-Zipf pour rendre compte des contributions multiples. Cette loi énonce que la fréquence des mots d'une langue naturelle est inversement proportionnelle à leur rang dans l'histogramme des mots par fréquences décroissantes. Autrement dit, si le mot le plus fréquent apparaît N fois, le k-ième mot le plus fréquent apparaît $\frac{N}{k}$ fois. Cette distribution, observée empiriquement sur divers documents ou ensembles de documents, a été démontrée par Mandelbrot [Mandelbrot, 1965], qui en a proposé une généralisation. La comparaison de distributions d'occurrences dans des documents avec la distribution théorique de Zipf donne ainsi un indice quant à la représentation dans le document de chaque phénomène avec une fréquence "naturelle" ou non. Dans la section suivante, nous comparons les distributions dans les corpus collectés ainsi que dans des corpus considérés de référence que nous présenterons. Nous y étudions notamment les effets, attendus ou non, des biais présents dans les corpus considérés.

2.5.2 Analyses

Les distributions des corpus en langue naturelle ont été calculées en prenant le token comme trait caractéristique. Cependant, il nous a semblé plus pertinent d'étudier les distributions des lignes de commandes dans les corpus en langage formel car c'est une unité sémantique identifiable. Nous avons suivi la règle utilisée pour calculer les commandes uniques dans le tableau 2.2 : les traits considérés sont formés des lignes complètes pour le corpus R, et des têtes des commandes seulement pour bash1 et

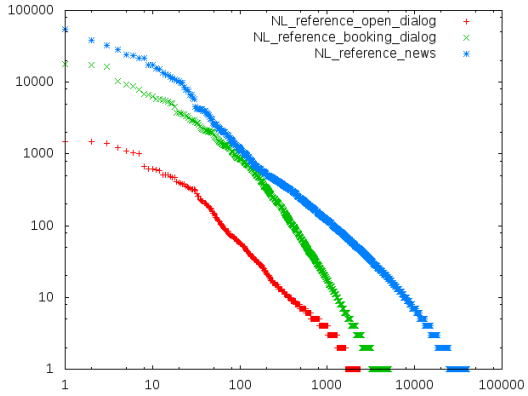


FIGURE 2.1 – Distribution des occurrences des unigrammes dans les corpus de référence pour la langue naturelle

bash2. Les distributions en tokens de ces demi-corpus ont quand même été calculées afin d'en estimer la différence avec une distribution naturelle. Quant aux demi-corpus de langue naturelle eux-mêmes, le trait retenu est la fréquence des unigrammes. Mesurer la fréquence des lignes n'est pas pertinent ici, car les demi-corpus étudiés et leurs corpus de référence associés n'ont pas la même structure.

Corpus monolingues de référence

La distribution des traits sur les parties des corpus en langue naturelle a été comparée avec les trois corpus de référence suivants :

- un corpus de dialogue humain-machine collecté avec le système orienté-tâche ARISE de recherche d'horaires de train et de réservation de billets [Lamel *et al.*, 2000] ($435K$ tokens)
- un corpus de dialogue humain-machine pour de la recherche interactive d'information précise en domaine ouvert provenant du projet RITEL [Rosset *et al.*, 2006b] ($41K$ tokens)
- un corpus d'émissions journalistiques radio et télédiffusées [Galliano *et al.*, 2005] ($1M$ de tokens)

Chacun d'eux est constitué de phrases courtes et les deux premiers proviennent d'interactions humain-machine. Ils présentent donc un rapport avec la tâche que nous étudions et devraient être comparables avec une liste de requêtes. Les distributions des tokens dans ces trois corpus sont présentées dans la figure 2.1 sur une échelle logarithmique en abscisses et en ordonnées. On remarquera qu'à cause des disparités en nombre de tokens entre les trois, seule la forme des courbes dessinées doit être utilisée pour comparaison, et non leur amplitude. On constate déjà que le corpus de recherche d'information et celui d'émissions journalistiques ont des distributions semblables, tandis que le corpus de réservation de billets contient un vocabulaire plus restreint en comparaison.

Concernant la partie du corpus en langage formel, des fichiers "`.bash_history`" ont été rassemblés à partir de dépôts en ligne librement accessibles [8] pour comparaison avec les parties en langage formel. Ce corpus contient un total de $44K$ tokens, pour $21K$ lignes de commande. Les figures 2.2 et 2.3 présentent respectivement la distribution des occurrences des têtes de commandes et

8. Il s'agissait essentiellement de dépôts github.

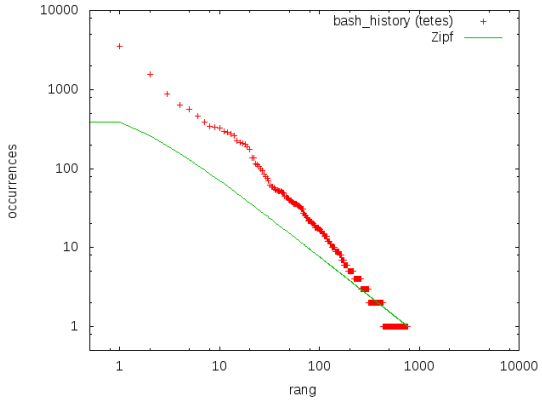


FIGURE 2.2 – Distribution des occurrences des commandes (têtes) dans le corpus de référence `bash_history`

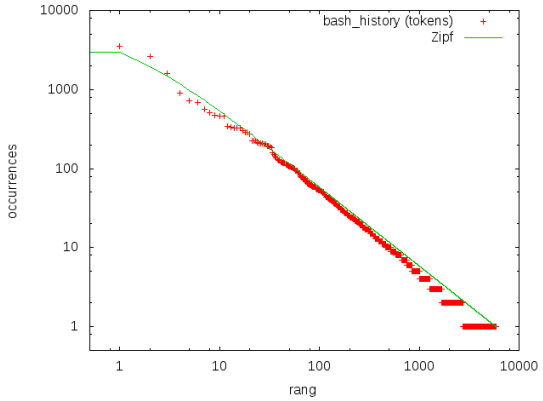


FIGURE 2.3 – Distribution des occurrences des tokens dans le corpus de référence `bash_history`

des tokens dans le corpus `bash_history`, des courbes de Zipf correspondantes[9] ont été ajoutées pour comparaison. Ces dernières ont été calculées sur la base du nombre total de symboles distincts présents dans l'histogramme : pour N symboles distincts, la courbe passera du point $(1, N)$ au point $\left(N, \frac{1}{N}\right)$. Notez que le plateau au tout début de ces courbes est trompeur puisque la loi de Zipf est une discrétisation de la fonction $\frac{1}{x}$. Pour cette même raison, aucun point de donnée ne peut être défini en deçà de l'abscisse 1, car il s'agit de rangs. Alors que la distribution des commandes montre un excès des lignes fréquentes par rapport à ce que la distribution de Zipf prédit, l'histogramme des tokens est en étonnante adéquation avec la courbe zipfienne. Cela semble indiquer que le principe d'économie, par lequel on explique la loi de Zipf, s'applique indifféremment à l'usage des tokens d'un langage formel ou des mots d'une langue naturelle.

La section suivante présente les résultats de nos analyses comparatives avec ces corpus de référence, ainsi que des éléments de discussion qu'ils soulèvent. Les courbes de Zipf ajoutées aux figures présentées dans la suite sont calculées selon le même principe que décrit ici.

Distributions comparées

Les figures 2.4 à 2.12 présentent les distributions respectives des demi-corpus en langage formel, puis en langue naturelle. Chacune d'elles représente l'histogramme des caractéristiques, tracé à l'échelle logarithmique en abscisses et en ordonnées. Une courbe normalisée de la loi de Zipf a été ajoutée sur les histogrammes qui représentent un seul corpus. Les courbes doivent être comparées selon leur forme plutôt que leur amplitude. En effet cette dernière dépend fortement de la taille des corpus, qui n'est pas toujours homogène. Toutes les courbes de cette section sont cependant présentées avec la même échelle.

Comme attendu, l'histogramme des commandes du corpus `R`, équilibré, est quasiment plat, (figure 2.4). Toute la redondance présente dans ce corpus a été ajoutée de manière tout à fait artificielle, afin d'enrichir les reformulations en langage naturel, sans modification des commandes. Les histogrammes concernant le demi-corpus en langage formel de `bash1` ont été reportés en figures 2.6 et 2.7. Ils sont difficilement exploitables en raison du faible nombre total de commandes, et dans une certaine mesure de tokens, dans le demi-corpus. Cependant, ils confirment bien que les variations de `bash1` sont fortement biaisées et comportent une redondance artificielle de même que dans le demi-corpus en langage formel de `R`. Le corpus `bash2` quant à lui (figures 2.8 et 2.9), montre une distribution des

9. Les courbes n'ont pas été discrétisées pour plus de lisibilité.

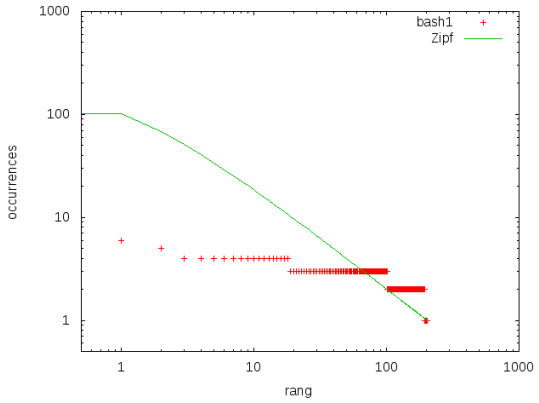


FIGURE 2.4 – Distribution des occurrences des commandes dans le corpus R

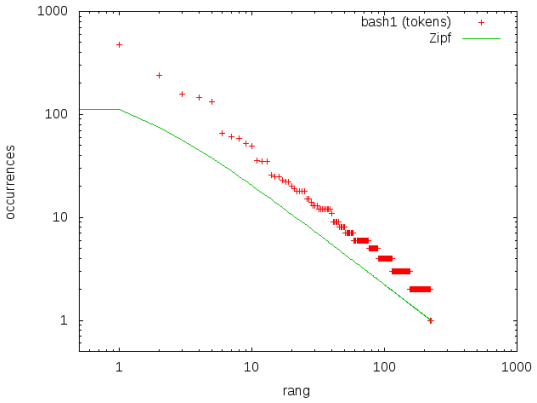


FIGURE 2.5 – Distribution des occurrences des tokens de la partie en langage formel dans le corpus R

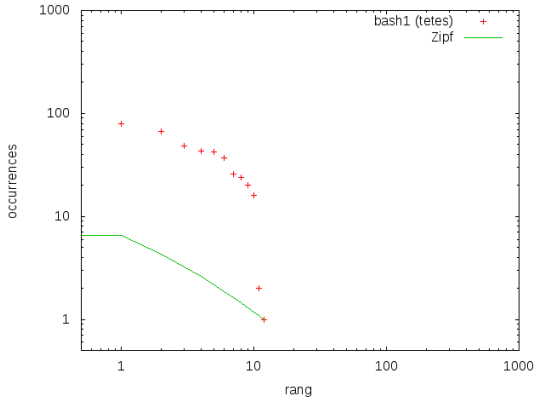


FIGURE 2.6 – Distribution des occurrences des commandes (têtes) dans le corpus bash1

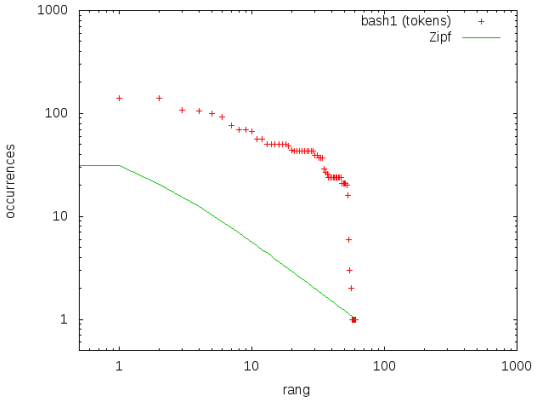


FIGURE 2.7 – Distribution des tokens de la partie en langage formel dans le corpus bash1

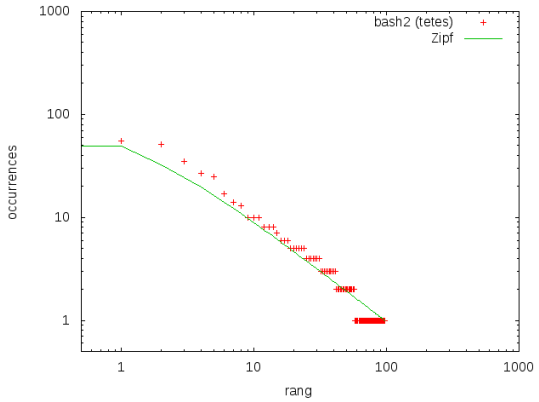


FIGURE 2.8 – Distribution des occurrences des commandes (têtes) dans le corpus bash2

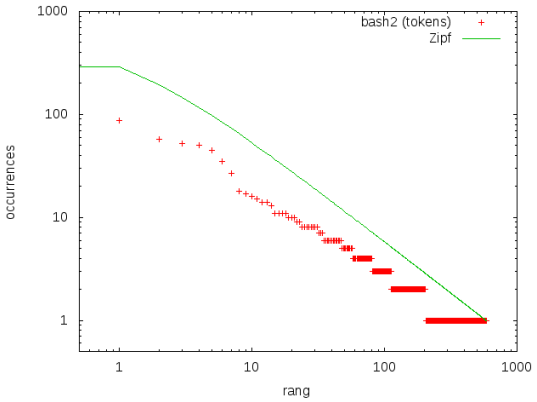


FIGURE 2.9 – Distribution des tokens de la partie en langage formel dans le corpus bash2

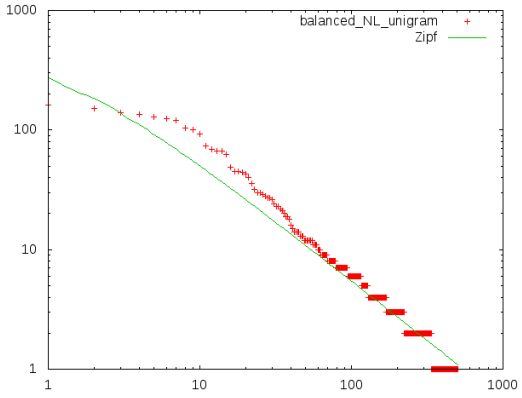


FIGURE 2.10 – Distribution des occurrences des unigrammes dans la partie en langue naturelle du corpus R

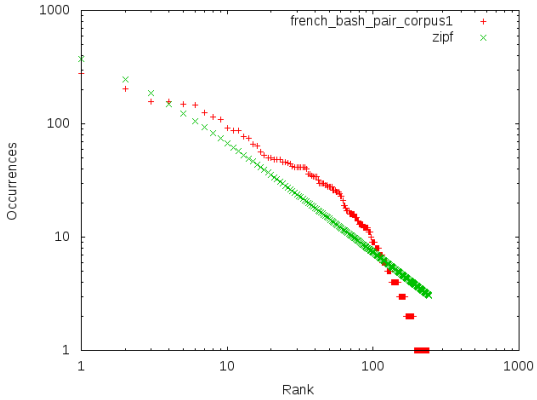


FIGURE 2.11 – Distribution des occurrences des unigrammes dans la partie en langue naturelle du corpus bash1

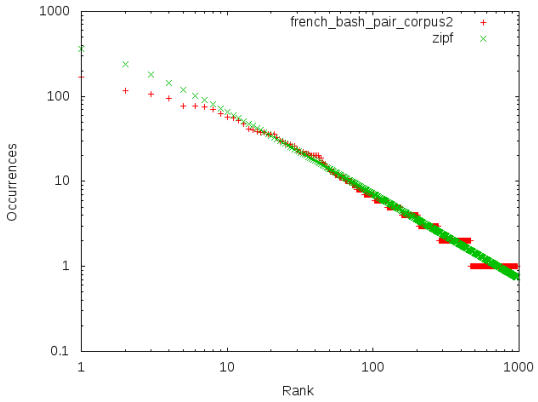


FIGURE 2.12 – Distribution des occurrences des unigrammes dans la partie en langue naturelle du corpus bash2

commandes beaucoup plus proche de la loi de Zipf et donc de ce à quoi on pourrait s'attendre de manière générale, venant d'utilisateurs en conditions non biaisées. La forme de l'histogramme est comparable avec celle du corpus de référence présenté sur la figure 2.2. Il semble donc que ce dernier corpus soit le plus représentatif des trois, en ce qui concerne sa distribution propre. Ce résultat ne nous permet pas d'en tirer une conclusion plus générale, car bien que les distributions soient proches en valeurs, rien n'indique que les mêmes commandes ont les mêmes fréquences dans l'une et dans l'autre.

La comparaison des histogrammes concernant les demi-corpus en langue naturelle est moins évidente (voir figures 2.10, 2.11 et 2.12). Cela est dû notamment aux différences non négligeables des distributions entre les corpus de référence eux-mêmes. Cependant, on peut identifier que les corpus `R` et `bash1` montrent un excès d'unigrammes de fréquence moyenne par rapport à la courbe de Zipf, ainsi qu'un peu moins d'unigrammes fréquents en comparaison. On peut l'expliquer pour le corpus `bash1` par le fait que les reformulations proposées par les participants à la collecte étaient pour partie fondées sur la reprise de segments communs, avec la variation d'autres sous-segments. Le lecteur peut consulter l'exemple 2.2 vu plus haut, pour un échantillon (non exhaustif) des types de reformu-

lations dans ce demi-corpus. Le même mécanisme apparaît dans le demi-corpus `R`. Bien qu'il n'y ait pas eu de phase de reformulation systématique des requêtes, le participant semble avoir implicitement équilibré le vocabulaire des expressions méta du type "donne-moi le résultat de", "retourne", "affiche" ou d'autres expressions à l'impératif synonymes. Cela explique également l'observation d'un déplacement des occurrences des unigrammes très fréquents vers les unigrammes de fréquences moyennes sur la figure 2.11, mais seulement une augmentation dans les fréquences moyennes sur la figure 2.10, car les requêtes du demi-corpus `R` n'ont pas été systématiquement dupliquées pour reformulation comme l'ont été la plupart de celles de `bash1`.

Enfin, l'histogramme du demi-corpus `bash2` (figure 2.12) semble correspondre particulièrement bien à une distribution de Zipf. On peut l'interpréter comme une bonne représentativité du corpus *sur son propre domaine*, bien que cela n'apporte finalement que peu d'information. Nous en discutons dans la section suivante.

2.6 Conclusion et discussion

Parmi les choix possibles quant au format de représentation des exemples du corpus, nous avons retenu le plus générique, qui est aussi le plus simple : l'association entre la chaîne de caractères représentant la requête, et la chaîne de caractères représentant la commande. Ce choix est motivé par le principe de parcimonie que nous appliquons à l'objectif, relativement nouveau, de transférer les énoncés en langue naturelle vers des commandes en langage formel de manière complètement générique. Ce principe conduit donc à privilégier en priorité l'approche la plus simple, pour en tirer des enseignements concernant les directions à suivre. Nous avons défini une **requête** comme un énoncé dans une langue naturelle donnée, et la **commande** associée comme l'expression dans un langage formel donné de la ligne d'instructions permettant d'obtenir le comportement décrit dans la requête. Les données que nous avons collectées sont constituées de requêtes en français, et de commandes en `R` pour l'un des corpus, et en `bash` pour les deux autres.

Les sources de données compatibles avec le format choisi ne sont pas nombreuses, et souvent bruitées. Nous avons organisé des collectes d'exemples afin d'obtenir des bases plus adaptées à la situation incrémentale, mais aussi moins bruitées et plus naturelles. Chacune des collectes a été menée avec une attention différente aux biais possibles, influençant la représentativité des corpus. Une bonne représentativité d'un corpus entraîne des occurrences plus élevées et variées des phénomènes courants et plus faibles pour les plus rares. Cela présente donc l'avantage de traiter plus efficacement ces phénomènes courants. En conséquence, le réalisme des mesures de performance du système de transfert doit s'en trouver amélioré. On peut également supposer que la performance réelle en cas de déploiement, sera aussi positivement influencée par l'emploi d'une base d'exemples plus représentative.

La réprésentativité d'une base d'associations entre requêtes et commandes n'est pas triviale, les mesures de représentativité proposées dans la littérature concernent des corpus de langue naturelle non contrainte et ne sont donc pas pertinentes pour les demi-corpus de requêtes et *a fortiori* pour les demi-corpus de commandes. Nous avons comparé nos corpus à des corpus de référence selon les distributions de leurs caractéristiques pour en estimer la représentativité. Nous les avons également comparé à la loi de Zipf, qui décrit le comportement statistique de la langue naturelle. La comparaison des bases collectées avec des corpus de référence montre des disparités dans les distributions de leurs caractéristiques. La distribution des commandes dans les demi-corpus en langage formel est plutôt éloignée de la référence pour les corpus `R` et `bash1`, mais beaucoup plus proche pour `bash2`. On retrouve moins de variation entre les distributions des unigrammes dans les demi-corpus en langue naturelle, que ce soit entre elles ou avec les trois corpus de référence. `bash2` montre une adéquation prononcée avec la distribution prévue par la loi de Zipf, qu'il s'agisse de son demi-corpus en langue naturelle ou de celui en langage formel. Cependant, l'interprétation de cette adéquation est à construire

avec précaution, car bien qu'il soit admis que cette loi prédit les distributions des caractéristiques de la langue pour un grand nombre d'observations, elle s'applique également à d'autres domaines tels que la distribution de la taille des mots pour une génération *aléatoire*. En outre, la figure 2.1 montre bien qu'entre plusieurs distributions naturelles, des différences notables sont observables dans la forme de la courbe obtenue. Cela diminue donc l'importance des différences comparées observées entre les différentes figures.

À l'aide de ces corpus collectés, nous allons à présent étudier les caractéristiques de différentes méthodes dites d'apprentissage appliquées au transfert de la langue naturelle vers le langage formel. Le chapitre suivant présente l'application préliminaire d'une méthode naïve pour le transfert, appliquée au corpus `R`. Puis, dans la seconde partie, nous explorons les méthodes de transfert fondées sur l'analogie formelle, que nous appliquons à tous les corpus collectés décrits ici, afin d'en comparer les caractéristiques à la lumière des performances qu'ils induisent.

Chapitre 3

Étude préliminaire

Sommaire

3.1 Introduction

La tâche d'interpréter des requêtes formulées en langage naturel pour produire des commandes correspondantes dans un langage formel peut être vue sous certains aspects comme une simplification de la traduction automatique entre deux langages naturels. En effet, d'une part le langage cible est beaucoup moins riche et complexe qu'un langage naturel, et d'autre part, le spectre des expressions correspondant à des requêtes est un sous-domaine spécifique du langage naturel. En revanche, le langage formel obéit à des règles strictes et non ambigües, ce qui ajoute une contrainte pour l'étape de production des commandes. En particulier, là où un lecteur humain comprendra un énoncé en langue naturelle approximatif voire incorrect, la machine ne peut interpréter ni exécuter une expression ambigüe ou syntaxiquement incorrecte.

Nous considérons dans ce chapitre une approche naïve pour cette tâche de transfert fondée sur l'association directe des requêtes avec les commandes et la similarité de surface entre requêtes. Le développement de cette approche est motivé par l'obtention d'une méthode de référence afin d'évaluer d'autres techniques de manière critique. De plus, compte tenu de la restriction du domaine, on peut s'attendre à ce qu'une approche par association directe donne un score de référence intéressant.

Les travaux existants proposant des approches pour des tâches similaires ou voisines sont présentés dans la section suivante. Nous détaillons notre approche par association directe et similarité dans la suite.

3.2 Approche naïve par association directe et similarité de surface

Les domaines source et cible du transfert sont très différents d'une étude à l'autre, comme on peut le lire en section 2.2. Notamment, certains travaux s'intéressent à des données structurées. Pour la source, il s'agit d'arbres d'analyse syntaxique, quant à la cible, il peut s'agir de représentations DOM [1], de séquences de commandes ou encore de graphes d'actions. En outre, les approches sont spécifiques au domaine source et/ou au domaine cible, rendant difficile la comparaison autre que qualitative de ces travaux. Afin de faciliter l'alignement avec d'autres types de données, mais aussi pour garantir la généricité du modèle pour tout type d'applications, nous nous sommes donnés la contrainte de l'indépendance des langages source et cible.

L'indépendance du langage cible implique que les réponses produites grâce au modèle proviennent nécessairement d'une transformation des connaissances acquises. Autrement dit, on ne dispose pas de règles explicites d'association entre objets du langage source et objets du langage cible. On ne dispose pas non plus de connaissances *a priori* permettant de déterminer la validité d'un énoncé du langage cible. La base de connaissances accessible prend la forme d'un ensemble d'associations liant un énoncé du langage source à un énoncé du langage cible. Toute règle relative au langage source, au langage cible ou aux associations entre chacun d'eux doit donc être inférée de cette base d'associations. La base d'exemples est l'ensemble :

$$B \in S \times C$$

S et C étant respectivement le langage source et le langage cible. Les projections respectives de B sur le langage source et le langage cible sont :

$$B_s = \{s | \exists c \, (s, c) \in B\}$$

$$B_c = \{c | \exists s \, (s, c) \in B\}$$

1. Document Object Model

	Requêtes	**Commandes (en R)**
1	Charge les données depuis **"res.csv"**	`var1 <- read.csv("res.csv")`
2	Trace un histogramme de la colonne **2** de **tab**	`plot(hist(tab[[2]]))`
3	Dessine la répartition de la colonne **3** de **tab**	`plot(hist(tab[[3]]))`
4	Trace la répartition de la colonne **3** de **tab**	`plot(hist(tab[[3]]))`
5	Somme les colonnes **3** et **4** de **tab**	`var2 <- tab[[3]] + tab[[4]]`
6	Somme les colonnes **3** et **4** de **tab**	`var3 <- sum(c(tab[[3]], tab[[4]]))`

TABLEAU 3.1 – Un échantillon d'associations entre énoncés en langue naturelle et commandes en langage formel. Chaque exemple comprend la formulation d'une requête et la commande attendue en réponse. Les éléments en gras sont les paramètres, appairés entre la requête et la commande. On peut noter qu'un énoncé peut correspondre à plusieurs commandes différentes et inversement.

Les langages source et cible sélectionnés étant respectivement composés d'énoncés en langue naturelle et de commandes, nous désignerons dans la suite les paires énoncé-commande par la notation (e, c).

Nous présentons ici une méthode générique naïve fondée sur la sélection d'un énoncé en langage cible directement dans B_c. Cette méthode repose sur l'identification de l'énoncé $s \in B_s$ le plus similaire à l'énoncé $e_{req} \in S$ à traduire. Les énoncés cibles associés $C = \{c | (e, c) \in B\}$ sont considérés comme les plus proches de la solution idéale dans la base, on les sélectionne alors pour répondre à la requête. Dans le cas de multiples énoncés cibles associés, il n'est pas possible de les discriminer. Seul l'un d'entre eux est alors sélectionné.

Nous discuterons des nombreuses limitations dont souffre cette approche dans le bilan de cette partie (page 63). Elle s'apparente en effet à un apprentissage par cœur des associations au cas par cas. Cependant, rappelons que le langage cible auquel nous nous intéressons est un langage formel. Il est par définition non ambigu et possède un vocabulaire très limité. Le nombre d'énoncés possibles est donc faible au regard des possibilités d'un langage naturel. Cela implique que le nombre de formulations en langage naturel pour une commande donnée est beaucoup plus grand que le nombre de formulations possibles de la commande en langage naturel. Remarquons qu'il ne s'agit pas pour autant d'une méthode *ad hoc* puisqu'aucune connaissance extérieure [2] dépendante du domaine n'est utilisée. On peut alors formuler l'hypothèse que de petites variations dans le lexique ou la formulation d'une requête en langage naturel conservent la sémantique globale de la requête et correspondent à la même commande en langage formel. Bien que cette hypothèse mérite d'être discutée, cela permet d'espérer de cette approche naïve un score non négligeable et de l'utiliser pour les comparaisons avec des méthodes plus fines.

La base d'exemples B représente le sous-ensemble connu de la relation binaire $\mathcal{R} : S \to C$ qui associe un énoncé en langue naturelle à une commande en langage formel. Remarquons d'abord que $\mathcal{R}$ n'est pas définie sur l'ensemble tout entier des énoncés du langage. En effet, les énoncés qui ne sont pas des requêtes ne sont associés à aucune commande, de même que les requêtes qui sont irréalisables à l'aide du langage formel sélectionné.

3.2.1 Topologie des associations

Dans le cas le plus simple, $\mathcal{R}$ est fonctionnelle (toute requête a au plus une image par $\mathcal{R}$) et injective (toute commande a au plus un antécédent par $\mathcal{R}$). Cependant il suffit de considérer l'exemple 3.1 pour se convaincre que ce n'est pas réaliste. En effet, plusieurs énoncés en langue naturelle peuvent être

2. On appelle connaissance extérieure toute information liée au domaine source ou cible qui ne peut pas être acquise par un incrément de la base d'exemples.

Depuis combien de temps la machine est-elle allumée ? Donne moi l'uptime. Quand était le dernier démarrage ? Affiche la durée de fonctionnement continu de l'ordinateur.

EXEMPLE 3.1 – Plusieurs alternatives pour demander l'exécution de la commande `system("uptime")` accédant au shell UNIX depuis `R` en langue naturelle.

synonymes et donc correspondre à la même commande. La relation n'est donc pas injective, plusieurs énoncés peuvent être associés à une seule et même commande.

D'autre part, les deux dernières associations du tableau 3.1 montrent qu'un unique énoncé peut, dans certains cas, être associé à plusieurs commandes différentes [3]. Le cas le plus fidèle à la réalité est donc celui d'une relation qui n'est ni injective ni fonctionnelle. Plusieurs énoncés peuvent être associés à la même commande, et un seul énoncé ambigu peut renvoyer à plusieurs commandes différentes (voir les exemples 5 et 6 du tableau 3.1). Nous devons considérer tous ces cas de figure pour comparer la requête à traduire avec les requêtes de B_s en vue de sélectionner la ou les commandes à retourner. On peut remarquer que la non injectivité et la non fonctionnalité de $\mathcal{R}$ ne contreviennent pas à la non ambiguïté du langage formel. En effet, un langage $\mathcal{L}$ est non ambigu si la relation entre les énoncés de $\mathcal{L}$ et les unités de sens est fonctionnelle. Cette relation est donc tout a fait distincte de $\mathcal{R}$.

3.2.2 Sélection

Plusieurs stratégies non exclusives peuvent être considérées pour sélectionner un énoncé cible à retourner pour le système à l'aide d'une mesure de similarité $\sigma : S \times S \to \mathbb{R}$ entre deux énoncés en langue naturelle. Étant donnée une liste de commandes triées par la similarité de leur antécédent en langue naturelle, il faut déterminer si le système doit répondre, et si oui, combien de commandes il doit retourner.

La première stratégie est axée sur le nombre de réponses données pour chaque énoncé-requête e_{req}. Les n premières commandes relativement au classement de leurs énoncés associés dans B sont retournées. Le rang $r(e|e_{req})$ d'un énoncé $e \in B_s$ par rapport à la requête e_{req} est donné par le nombre d'énoncés de B_s dont la similarité avec e_{req} est supérieure à celle de e avec e_{req}.

$$r(e|e_{req}) = \left|\left\{e' \in B_s : \sigma(e_{req}, e') > \sigma(e_{req}, e)\right\}\right|$$

Le second choix de stratégie est de déterminer un seuil de similarité en dessous duquel les énoncés candidats de B et leurs commandes associées sont considérés trop différents pour correspondre. Cela permet de choisir si une réponse est donnée ou non, et donc d'autoriser le silence du système, mais n'offre pas de contrôle sur le nombre de commandes retournées (bien qu'on puisse le conserver par la suite sous un seuil raisonnable). Cette stratégie retourne donc comme résultat l'ensemble des commandes dont l'énoncé associé dans B a une valeur de similarité avec e_{req} supérieur au seuil déterminé :

$$Res = \{c : (e, c) \in B, \sigma(e_{req}, e) > s\}$$

avec s le seuil de similarité sélectionné. Le tableau 3.2 illustre le classement des énoncés de B par similarité décroissante avec e_{req}. Notons que, bien que les énoncés de la base de connaissances aient été ordonnés, rien ne nous permet de discriminer les commandes qui leur sont associées. En conséquence, l'application de la première stratégie avec un nombre de commandes égal à 4 sélectionne les

3. Ici, le sens des deux commandes est réellement différent. La première calcule la somme ligne par ligne tandis que la seconde calcule la somme totale de toutes les valeurs.

deux commandes associées à e_1, et un sous-ensemble quelconque de taille 2 parmi les commandes associées à e_4. Afin de rationaliser cette sélection, on peut envisager de pondérer chaque association

e	$\sigma(e, e_{req})$	commandes associées à e dans B
e_1	$0,80$	$\{c_2, c_6\}$
e_4	$0,74$	$\{c_4, c_5, c_7\}$
e_3	$0,36$	$\{c_3, c_6\}$
e_2	$0,36$	$\{c_1\}$

TABLEAU 3.2 – Représentation d'un classement possible des énoncés e_1 à e_4 par rapport à un e_{req} donné.

de B en fonction du nombre d'utilisations correctes et incorrectes (par exemple l'association $e_1 \rightarrow c_6$ reçoit un poids de 4 car la commande c_6 a été retournée pour e_1 cinq fois avec succès et une fois à tort). Ainsi, les commandes les moins sujettes à erreurs seront toujours privilégiées. Cependant, il ne s'agit que d'une optimisation de surface et elle demande des informations sur l'utilisation du système, ce dont nous ne disposons pas. D'autre part, on peut également remarquer qu'une commande peut apparaître à plusieurs endroits dans le classement des énoncés de B. En effet, c_6 apparaît pour e_1 et pour e_3, avec deux valeurs de similarité différentes. Dans ce cas, seule la mieux classée sera retenue.

La formulation proposée du problème repose sur la fonction de similarité σ et sur les associations présentes dans la base de connaissances. Nous allons maintenant présenter notre approche pour établir ces associations et les fonctions de similarité que nous y appliquons.

3.2.3 Description de l'approche

Cette section décrit les étapes constitutives de l'approche fondée sur la similarité des énoncés que nous proposons. Initialement, un algorithme hors ligne prétraite la base d'associations afin de rendre possible (et pertinente) l'indexation des termes. Puis, lors de l'interrogation du système, les énoncés sont classés par ordre de proximité décroissante avec la requête de l'utilisateur à l'aide de plusieurs mesures de similarité.

Constitution de la base indexable L'association correcte entre requêtes et commandes requiert au moins la prise en compte de leurs paramètres respectifs (noms de variables, valeurs numériques et chaînes de caractères entre guillemets). Les représentations génériques des énoncés et des commandes sont construites à l'aide de l'identification des paramètres dans le couple à intégrer à la base de connaissances (voir tableau 3.1). Ces représentations sont utilisées par la suite pour reconstruire la commande à l'aide des paramètres de l'énoncé-requête.

La base de connaissances ne contient que les formes génériques des commandes. Il s'agit du texte de la commande comportant des références non résolues pour chaque paramètre qui a été associé à un élément de l'énoncé d'apprentissage. Ces références sont résolues à la phase de recherche par association avec les tokens correspondants de l'énoncé-requête.

Similarités entre énoncés Nous avons appliqué trois mesures de similarité différentes pour la recherche de commandes afin de comparer leurs influences respectives sur le comportement du système : l'indice de Jaccard, une aggrégation du TF-IDF [4], ainsi que le score BLEU [5]. Le choix de ces trois méthodes est dicté par la diversité des caractéristiques qu'elles mesurent, qui provient elle-même des cadres dans lesquels elles ont chacune été développées.

4. Term Frequency-Inverse Document Frequency
5. Bilingual evaluation understudy

Indice de Jaccard L'indice de Jaccard mesure la similarité entre deux ensembles à valeurs dans le même domaine. Il s'agit initialement d'une mesure de distance entre ensembles [Kosub, 2016], dont le complément donne donc une proximité. Dans notre cas, nous comparons l'ensemble des mots de la requête de l'utilisateur en langue naturelle et celui de chacune des requêtes présentes dans la base. Ils sont chacun valués dans l'ensemble des tokens possibles. L'expression de l'indice de Jaccard adaptée pour mesurer la similarité de deux énoncés e_1 et e_2 est :

$$J(e_1, e_2) = 1 - \frac{|M(e_1) \cap M(e_2)|}{|M(e_1) \cup M(e_2)|}$$

où $M(e)$ désigne l'ensemble des mots de l'énoncé e utilisés. On notera le complément à 1, permettant de passer d'un classement par distances décroissantes à un classement par similarités décroissantes. En utilisant cette mesure, il semble pertinent d'ignorer les mots outils dans les énoncés. En effet, la comparaison des énoncés sous la forme d'ensembles de mots fait perdre toute information sur leur ordre dans la phrase. Sans le contexte, les mots outils n'apportent que peu d'information et introduisent plutôt un biais dans le ratio des nombres de tokens. L'indice de Jaccard est une méthode standard pour évaluer les co-occurrences des unigrammes. Elle devrait être plus efficace avec des données comprenant peu d'exemples ambigus en termes de vocabulaire.

TF-IDF La mesure TF-IDF calcule la représentativité d'un mot dans un document, par rapport à un corpus. Elle permet donc, étant donné un mot, de classer les documents du corpus selon sa représentativité pour chacun d'eux. L'avantage de TF-IDF est qu'il tient compte de la fréquence du mot dans le corpus. Ainsi, un mot apparaissant dans tous les documents, même s'il est très fréquent, ne sera représentatif d'aucun document. Il existe de multiples manières de calculer le TF-IDF d'un mot par rapport à un document, voir [Manning *et al.*, 2008b]. Dans notre cas, nous calculons cette mesure à l'aide des fréquences brutes, comme le décrit la formule donnée plus bas. Il faut ici évaluer la représentativité d'une phrase plutôt que d'un seul mot. On utilise donc une aggrégation des valeurs de TF-IDF pour chacun des mots composant l'énoncé-requête en langue naturelle.

$$tfidf_e(e_{req}, e_{comp}) = \frac{1}{|M\,(e_{req})|} \sum_{m \in M(e_{req})} tfidf(m, e_{comp}, E)$$

avec $E = \{enonce | (enonce, commande) \in B\}$ l'ensemble des énoncés de la base de connaissances, et où e_{req} est l'énoncé-requête et e_{comp}, l'énoncé comparé. La fonction $tfidf$ elle-même s'exprime par :

$$tfidf(m, e_{comp}, E) = \frac{|\{m_e | m_e \in e_{comp} \wedge m_e = m\}|}{|\{m_e | m_e \in e_{comp}\}|} \times log\left(\frac{|E|}{|\{e \in E | m \in e\}|}\right)$$

Cette mesure ne nécessite plus la restriction aux mots pleins car TF-IDF inclut déjà la normalisation par rapport à la fréquence globale (dans le corpus) des termes.

Le score BLEU Le score BLEU [Papineni *et al.*, 2002] est une méthode de calcul de similarité qui a été développée pour automatiser l'évaluation de la traduction automatique. Il a été défini pour représenter une mesure de proximité avec l'évaluation par l'humain et est donc pertinent pour la recherche de paraphrases. Cette méthode s'appuie sur la mesure de co-occurrences entre n-grammes et permet de distinguer les énoncés candidats dans lesquels l'ordre des mots est trop différent de celui dans la référence (énoncé-requête). La valeur de précision modifiée qu'elle introduit est basée sur le rapport des co-occurrences de n-grammes entre candidat et référence, sur la taille totale du candidat, normalisée selon n.

```
<_operation>
  <_action> charge|_~V </_action>
  <_det> les </_det>
  <_subs> données|_~N </_subs>
  <_prep> depuis </_prep>
  <_mot_inconnu>
    "res.csv"
  </_mot_inconnu>
</_operation>
```

EXEMPLE 3.2 – Résultat de l'analyse par le système WMATCH de la requête "charge les données depuis "res.csv" "

$$P_{BLEU} = \sum_{gram_n \in e_{req}} \frac{max_{e_{comp} \in E} occ(gram_n, e_{comp})}{|\{gram_n \in e_{req}\}|}$$

où $occ(gram_n, e) = \sum_{gram'_n \in e}[gram_n = gram'_n]$ est le nombre d'occurrences du n-gramme $gram_n$ dans l'énoncé e. On peut noter que le dénominateur, qui correspond au nombre de n-grammes de l'énoncé e_{req}, peut aussi s'écrire $|e_{req}| - (n-1)$. La méthode de calcul du score BLEU comprend également une pénalité pour la brièveté, et ainsi empêcher les longs énoncés d'être trop désavantagés. Cependant, les énoncés comparés dans notre cas ne sont pas des documents de longueur très variables mais des expressions opérationnelles de tailles et d'amplitudes beaucoup plus raisonnables. Nous avons donc laissé de côté cette normalisation.

3.2.4 Prétraitements syntaxiques

Avant d'évaluer leur similarité, nous avons appliqué plusieurs combinaisons de filtres sur les tokens des énoncés à prendre en compte. Ces filtres sont appliqués à partir de l'analyse produite par le système WMATCH et présentée dans l'exemple 3.2. Il est possible de conserver ou non les mots outils ou les éléments non lexicaux. Ces derniers regroupent les valeurs numériques, les chaînes de caractères entre guillemets et les noms de variables. Les noms de variables sont les mots inconnus de l'analyseur syntaxique qui correspondent à une sous-partie identique dans la commande. Si les éléments non lexicaux sont conservés, ils sont transformés en substituts standardisés. De plus, il est possible d'appliquer ou non la lemmatisation des termes lexicaux. Par exemple, en ignorant les mots outils, en conservant les éléments non lexicaux et après application de la lemmatisation, le deuxième énoncé du tableau 3.1 serait transformé comme suit :

Trace un histogramme de la colonne **2** de **tab**

↓

TRACER HISTOGRAMME COLONNE **XXVALXX XXVARXX**

3.3 Application

3.3.1 Comparaison des mesures de similarité

Comme le montre le tableau 3.3 la mesure de similarité basée sur le TF-IDF domine les deux autres mesures en comparaison individuelle, quelle que soit la combinaison de paramètres de filtrage. En

Connaissances	Copie a.tar.gz vers b.tar.gz	`cp a.tar.gz b.tar.gz`
	Copie a.txt vers b.txt en mode interactif	`cp -i a.txt b.txt`
	Déplace a.tar.gz vers b.tar.gz	`mv a.tar.gz b.tar.gz`
Requête	Déplace a.txt vers b.txt en mode interactif	?

EXEMPLE 3.3 – Type de requête non résolue à l'aide de la recherche par similarité de surface pour les connaissances disponibles, résolue par analogie. Selon la mesure de similarité, une méthode par similarité de surface proposera l'une des commandes présentes dans les connaissances, ce qui ne peut pas correspondre à la solution.

effet, la forme des énoncés présents dans le corpus cause la répétition des mots du vocabulaire de l'interaction opérationnelle (verbes : "donner", "calculer", "afficher" ; noms : "vecteur", "matrice", "histogramme", ...). Cette structure peut expliquer que la fréquence inverse dans les documents (idf) soit particulièrement pertinente.

mots outils	inclus				non inclus			
non lexicaux	inclus		non inclus		inclus		non inclus	
lemmatisation	oui	non	oui	non	oui	non	oui	non
Jaccard	38,5%	34,6%	21,2%	23,1%	36,5%	36,5%	21,2%	23,0%
TF-IDF	**50%**	**50%**	**36,5%**	**40,4%**	**48,0%**	**51,9%**	**36,5%**	**40,4%**
BLEU	34,6%	34,6%	26,9%	30,8%	30,8%	32,7%	26,9%	30,8%
hasard	1,9%							

TABLEAU 3.3 – Mesures de précision par énoncé (P_e), en fournissant 3 réponses pour chaque énoncé requête test. L'ensemble des associations connues contient 85% du corpus, et l'ensemble de test 10%.

La lemmatisation et l'inclusion des mots outils ne semblent pas avoir une influence évidente sur la précision par énoncé. À l'inverse, on constate une amélioration dans tous les cas avec l'inclusion des éléments non lexicaux. Ce comportement provient au moins en partie de la longueur des énoncés de B ($7,5$ mots en moyenne), dont certains sont assez courts pour ne contenir aucun token lexical significatif (*"ln de A"*). L'analyse linguistique plus approfondie devrait permettre, notamment pour les mots outils, d'améliorer les performances par une exploitation plus fine de la structure des énoncés. Cependant, comme précisé en introduction, nous nous sommes fixé l'objectif d'indépendance à la langue ; l'utilisation de règles d'analyse fixes ne va pas dans ce sens.

La figure 3.1a montre la précision obtenue avec TF-IDF en fonction du nombre de commandes retournées pour chaque énoncé requête. D'après le graphique, il est intéressant de proposer au moins trois commandes à l'utilisateur afin d'obtenir une précision supérieure à 50%. En revanche, proposer à l'utilisateur un choix de plus de 5 éléments ne serait pas pertinent : le gain en précision par énoncé ne serait plus suffisant par rapport au temps supplémentaire nécessaire à l'utilisateur pour retrouver la bonne réponse (le cas échéant) parmi les propositions.

3.3.2 Autoriser le silence

Afin d'autoriser le système à ne pas toujours donner une réponse, nous avons fixé un seuil absolu pour la valeur de similarité des commandes à retourner. Les résultats montrent, quelle que soit la mesure de similarité choisie, qu'au moins les 6 seuils les plus sélectifs donnent toujours lieu à des réponses fausses. Ce résultat peut être dû à l'existence dans l'ensemble de test, d'exemples non couverts

par l'ensemble d'apprentissage. Plus vraisemblablement, il doit s'agir de commandes trop courtes (au plus un ou deux tokens lexicaux) pour être traitées correctement par la fonction de similarité.

3.3.3 Combinaisons

Une fois que chacune des méthodes proposées a été testée indépendamment, il peut être intéressant de tenter de les combiner. Cela permet notamment d'étudier leur complémentarité. L'oracle des 6 meilleures méthodes [6] montre une marge de progression intéressante (*cf.* figure 3.1b). Les résultats de la combinaison par le vote dépassent la meilleure méthode seule pour un nombre d'alternatives retournées inférieur à 4. Le vote atteint notamment 50% de bonnes réponses pour seulement 2 alternatives. La position de la courbe est moins claire pour un nombre d'alternatives plus élevé, mais notre problématique concerne l'utilisabilité du système, c'est pourquoi on s'intéresse plutôt aux gains pour un faible nombre de propositions. Cependant, il est indispensable d'effectuer les tests sur d'autres tirages de B dans le corpus afin d'obtenir une mesure de la variabilité de ces résultats.

On peut également exploiter la complémentarité des méthodes par l'entraînement d'un modèle d'apprentissage artificiel pour combiner *a posteriori* leurs résultats. La régression logistique n'est pas directement applicable à notre cas puisque les listes de commandes retournées avec les différentes mesures de similarité sont discrètes ; c'est-à-dire qu'on ne peut pas les fusionner directement à l'aide d'un paramètre réel. Nous avons donc utilisé la classification afin d'identifier les schémas pour lesquels l'une des méthodes est meilleure que les autres. Il s'agit d'entraîner un système à reconnaître les méthodes auxquelles se fier en fonction des indices dont nous disposons. Les valeurs de similarité discrétisées ont été employées en tant qu'attributs et l'ensemble des mesures ayant donné la bonne réponse en tant qu'étiquettes de référence. Ces paramètres ont été testés à l'aide des machines à vecteur de support de `libsvm` [Chang et Lin, 2011] avec un noyau polynomial de degré 3 et à coefficient de degré 0 nul. Les résultats sont présentés sur la figure 3.1b. Comme on pouvait s'y attendre, l'entraînement sur le petit corpus dont nous disposons ne produit pas d'excellents résultats. La performance de la prédiction n'atteint pas 20%, et le modèle détermine seulement la distinction entre les cas où il faut choisir la meilleure méthode et les cas où il vaut mieux ne pas répondre (dû à l'étiquette de classe "aucune"). Les résultats des tests avec le modèle appris parviennent à dépasser légèrement TF-IDF à partir de 5 réponses données, mais cela ne traduit pas une amélioration : d'une part, ce score est certainement obtenu grâce à l'abstention de réponse, qui est rendue possible par la présence de l'étiquette "aucune" lors de l'apprentissage, et d'autre part un tel nombre de réponses est déjà élevé si l'on considère que chacune des requêtes de l'utilisateur conduit à un choix multiple de 5 éléments.

3.4 Conclusion

3.4.1 Bilan

Nous avons appliqué trois méthodes de calcul de similarité pour le transfert de requêtes en langue naturelle vers des commandes en langage formel. Les résultats en termes de précision par énoncé ont atteint 60% de bonnes réponses après entraînement sur un petit corpus, tout en conservant le nombre d'alternatives de réponses proposées à l'utilisateur en deçà d'un seuil raisonnable (< 5 possibilités). Ces résultats vont dans le sens de notre hypothèse : les méthodes naïves de sélection permettent d'obtenir des résultats non négligeables lorsque le langage cible est un langage formel. Cependant, nous avons dû relâcher la contrainte de généricité fixée en introduction : l'utilisation de la lemmatisation par WMATCH fait en effet intervenir des connaissances extérieures à la base d'exemples concernant les flexions des mots d'une langue spécifique. Dans notre cas, il s'agit du français.

6. L'oracle donne toujours la bonne réponse si l'une au moins des méthodes considérées la fournit.

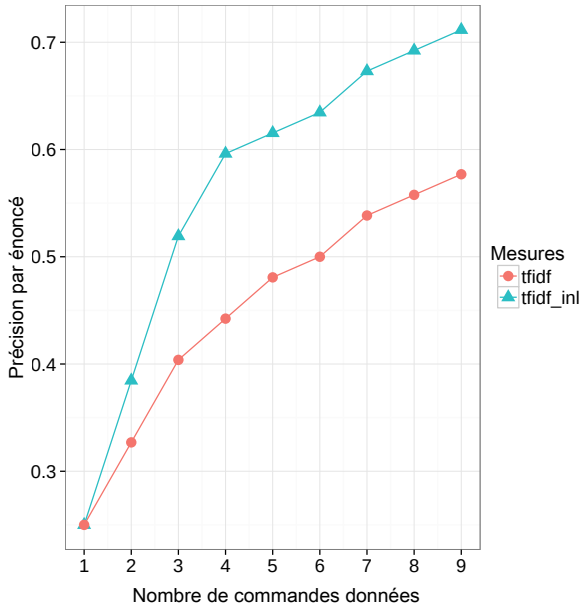
Beaucoup de méthodes approfondies ou d'approches parallèles peuvent être considérées pour optimiser le score obtenu, comme l'ajout ou le développement de mesures de similarité plus adaptées [Achananuparp *et al.*, 2008], la combinaison de l'apprentissage artificiel et du vote, ou encore l'entraînement d'un modèle pour ordonner ou réordonner les listes d'énoncés similaires. Cependant, ces méthodes demandent de grands volumes de données pour être efficaces. Elles sont également plutôt opaques en ce qu'elles ne permettent pas à l'utilisateur de comprendre facilement le raisonnement suivi pour proposer une réponse. De plus, la non fonctionnalité de la relation entre énoncés et commandes introduit des ambiguïtés impossible à résoudre à partir de la seule base de connaissances.

3.4.2 Autre application à l'approche par similarité

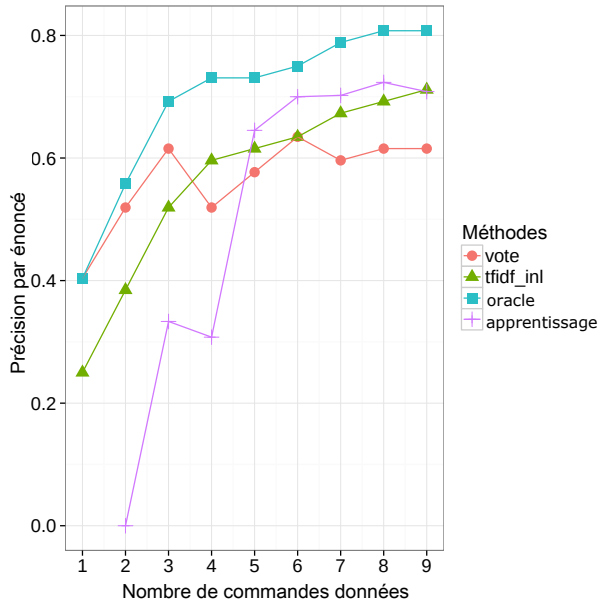
L'approche de sélection par TF-IDF que nous avons décrite dans ce chapitre a également été adaptée et appliquée dans le cadre du projet JOKER pour la sélection d'énoncés réponses lors d'un dialogue humain-machine [Duplessis *et al.*, 2016, Dubuisson Duplessis *et al.*, 2016]. L'application consiste à rechercher dans un corpus de sous-titres de films et de séries télévisées un énoncé susceptible de constituer une réponse pertinente à ce que l'utilisateur vient d'écrire. Pour cela, on représente le corpus comme un ensemble d'associations entre un énoncé dit initiative et un énoncé réponse, il s'agit de toutes les paires consécutives de sous-titres du corpus. Étant donné un énoncé écrit par l'utilisateur, on utilise la distance TF-IDF pour identifier l'énoncé initiative le plus proche de ce qui vient d'être écrit afin de proposer l'énoncé réponse associé pour poursuivre le dialogue.

Cette approche repose sur l'hypothèse que les paires d'énoncés constitués à l'aide du corpus de sous-titres correspondent bien à une initiative et une réponse. Nous ne discuterons pas ici de cette hypothèse. Les résultats obtenus grâce au système de dialogue décrit sont mitigés. Il peut s'agir d'un indice montrant une limite de l'approche TF-IDF, cependant de nombreux autres facteurs sont en jeu, tels que la qualité du corpus ou de son prétraitement ou l'accord inter-annotateurs lors de l'évaluation. Nous renvoyons le lecteur intéressé vers les publications associées pour plus de détails.

Au vu des résultats obtenus avec l'approche par similarité, nous proposons dans le bilan de cette partie une discussion des possibilités et limites de cette approche dans le cas général. Cette discussion introduira également la deuxième partie du document en proposant d'affiner la problématique établie dans l'introduction.



(a) Performances du système en utilisant la similarité TF-IDF avec et sans **i**nclusion des tokens **n**on **l**exicaux.



(b) Comparaison des performances pour les différentes combinaisons de méthodes. Les 6 méthodes combinées sont TF-IDF, Jaccard, BLEU, et ces 3 mêmes en **i**ncluant les tokens **n**on **l**exicaux.

FIGURE 3.1 – Précision par énoncé en fonction du nombre d'alternatives pour différentes méthodes. La précision par énoncé donne le nombre d'énoncés test pour lesquels au moins une des réponses proposées est correcte ; par cette définition, sa valeur ne peut donc qu'augmenter avec le nombre de réponses apportées pour chaque énoncé.

Bilan et nouvel axe de la problématique

L'application d'une méthode de sélection pour le transfert depuis la langue naturelle vers le langage formel a produit un système capable de donner une réponse correcte dans un peu plus de la moitié des cas. Ce résultat *a priori* satisfaisant masque cependant quatre aspects problématiques pour une application en contexte réel.

En premier lieu le score obtenu correspond au cas où le système donne trois réponses pour chaque requête. Cela suppose de la part de l'utilisateur qu'il ait la compétence d'identifier parmi celles-ci la commande correcte s'il y en a une, ainsi que la patience de toujours examiner trois commandes pour chacune de ses requêtes. Outre la possibilité de cas dans lesquels l'utilisateur ne sait pas choisir, cette multiplicité systématique accroît le risque d'erreurs de choix après une utilisation prolongée, et donc répétitive. Il serait appréciable que le système puisse déterminer de manière autonome, au moins dans une majorité de cas, la meilleure de ses propositions. Le problème de la patience de l'utilisateur se pose non pas par rapport aux performances théoriques du système, mais par rapport à son utilisabilité.

Le deuxième aspect concerne le problème de la confusion qui accompagne l'utilisation d'une mesure de similarité. Les résultats proposés en sont peu impactés du fait de l'homogénéité relative de l'ensemble d'entraînement et de l'ensemble de test pour le corpus R utilisé. Cependant, une approche centrée sur la proximité lexicale entre la requête de l'utilisateur et les requêtes présentes dans la base présente le désavantage de ne pas tenir compte de l'ordre des termes (ou des n-grammes, dans une moindre mesure). L'exemple 4 illustre une classe de confusion possible due à cette limitation. Ce cas de confusion semble cependant plutôt rare, et l'augmentation de la taille de n-grammes considérés ou la prise en compte des mots vides permettrait d'en éviter une partie.

Le troisième aspect à noter concerne l'identification et le remplacement adéquat des paramètres dans les requêtes et les commandes. Telle que décrite dans ce chapitre, l'identification des paramètres s'appuie sur l'analyse syntaxique de la requête, et leur remplacement suit une règle simple de positionnement linéaire. Par ailleurs, les paramètres non détectés sont inclus dans les critères de recherche d'une requête similaire. Encore une fois, ces critères ont probablement été surévalués dans ces expériences, car la base d'associations ainsi que la base de test comprenaient en grande partie des paramètres semblables et non lexicaux. En considérant un corpus plus varié, on peut s'attendre à des cas problématiques, voir l'exemple 5.

Enfin, le dernier aspect problématique est celui de la généralisation du système, qui est relativement faible. En effet, exceptée la détection et le remplacement des paramètres (qui est déjà sujet problématique), le système proposé n'est capable de produire aucune commande nouvelle, c'est-à-dire dont la structure syntaxique n'existe pas préalablement dans la base d'exemples.

copie la variable foo dans foo_copie et efface bar	`foo_copie = foo; bar = NULL`
efface la variable foo et la copie de bar	`foo = NULL; bar_copie = NULL`

EXEMPLE 4 – Deux associations aux commandes distinctes dont les requêtes ne diffèrent que de 3 tokens et sont donc susceptibles d'être confondues par une mesure de similarité fondée sur les tokens.

copie exacte dans variable	`variable = exacte`
crée un vecteur séquence partant de 1 en allant jusqu'à 5	`seq(1, 5)`
crée un vecteur séquence jusqu'à 5 en partant de 1	`seq(1, 5)`

EXEMPLE 5 – Associations mettant en évidence la difficulté de détection et/ou d'ordonnancement des paramètres des requêtes.

En conclusion, malgré les performances correctes obtenues avec l'approche proposée, on s'attend à les voir se dégrader avec les conditions plus variées d'une utilisation en contexte réel. Cette dégradation attendue est essentiellement due à l'utilisation de techniques *ad hoc* pour la recherche et l'adaptation de solutions. Il nous faut donc considérer la question de la généricité de l'approche comme l'élément central de la problématique. Un examen du corpus nous a permis de faire apparaître des régularités dans les formulations qui sont indépendantes du vocabulaire. Un sous-ensemble de la base d'exemples est présentée dans le tableau 4. On constate intuitivement que, disposant de l'exemple 4 dans les connaissances, il serait aisé de trouver une solution pour la requête "reste de la division de 1591 par 7" sans même analyser la structure linguistique de la phrase. De même, devant la requête à traiter "calcule le module du complexe Z", on conçoit assez facilement que sa structure peut être retrouvée dans les requêtes des exemples 1, 2 et 3. La commande solution "`Mod(Z)`" est dans ce cas déduite des commandes associées de ces exemples.

Ce mode de raisonnement est appelé *raisonnement analogique*, nous l'utilisons naturellement pour accomplir toutes sortes de tâches. Il a par ailleurs été étudié par la communauté des sciences de l'information et de la connaissance donnant lieu à une formalisation précise de son processus. Dans la seconde partie, nous proposons d'explorer l'application du raisonnement par analogie au problème de transfert depuis une requête vers une commande avec l'objectif de rendre notre approche complètement indépendante des langages utilisés.

1	donne la partie réelle de Z	`Re(Z)`
2	calcule la partie réelle du complexe Z	`Re(Z)`
3	donne le module de Z	`Mod(Z)`
4	reste de la division de X par 2	`X %% 2`

TABLEAU 4 – Extrait du corpus d'associations français - `R`

Deuxième partie

L’analogie comme moteur d’inférence incrémental

Chapitre 4

Raisonnement par analogie formelle

Sommaire

4.1 Introduction

Le raisonnement par analogie est abordé dans divers domaines autour de la cognition, la psychologie et de l'intelligence artificielle. Il est fréquemment présenté comme une fonction centrale dans la pensée humaine [Gentner *et al.*, 2001, Hofstadter et Sander, 2013]. Nous l'utilisons en effet volontiers plutôt qu'un raisonnement déductif pour résoudre des problèmes à partir d'exemples déjà résolus : rédiger une lettre à partir d'un modèle, construire une phrase dans une langue étrangère à partir de constructions connues, prédire le déroulé d'un roman ou d'un film à partir de schémas scénaristiques récurrents, imaginer la réaction d'une personne dans une situation par rapport à d'autres réactions observées... Pour cette raison, ce type de raisonnement est un candidat intéressant pour le développement d'approches en intelligence artificielle. En effet, le raisonnement mené par un système artificiel sur le principe de l'analogie est intelligible par l'humain grâce à sa familiarité, comme nous venons de l'évoquer. L'humain peut alors facilement décider d'approuver ou non une action lorsqu'il s'agit de l'utilisateur, ou bien remonter à la source théorique d'un comportement indésirable lorsqu'il s'agit du chercheur/développeur. D'autre part, et également en conséquence de sa proximité avec notre pensée, le raisonnement analogique est adapté au traitement du langage naturel, dont nombre de structures sont construites par analogie [Saussure, 1916, p. 221].

La notion d'analogie est employée selon des acceptions différentes, en linguistique [Saussure, 1916, Albright et Hayes, 2003, Boyé, 2016], en sciences cognitives [Gentner, 1983, Hofstadter *et al.*, 1994, Marshall, 2002, Gentner *et al.*, 2001, Hofstadter et Sander, 2013] et en intelligence artificielle [Prade et Richard, 2014]. Quant à son utilisation dans le langage courant, elle peut relever selon les cas de chacune de ces acceptions. On emploie également l'adjectif comparatif "analogue" comme synonyme de "semblable", "similaire", référant souvent à la fonction ou à la configuration des substantifs ainsi qualifiés. Toutes ces acceptions partagent la notion de transfert de propriétés depuis un domaine source vers un domaine cible.

Afin de mettre en œuvre le raisonnement analogique pour le transfert de langues, nous nous intéresserons essentiellement aux travaux proposant un cadre formel de l'analogie, permettant de reconnaître mais aussi de calculer des analogies. Les travaux de Hesse [Hesse, 1959] sont parmi les premiers sur cette voie, suivis beaucoup plus récemment par [Lepage, 1998] d'une part, et [Stroppa et Yvon, 2006] d'autre part. Nous adoptons dans la suite le cadre proposé par Nicolas Stroppa et François Yvon.

Une proportion analogique est une relation 4-aire notée $[\, x \,:\, y \,::\, z \,:\, t \,]$. Cette notation peut s'exprimer en français par "x est à y ce que z est à t". Deux opérations principales sont applicables à l'analogie proportionnelle selon le cas :

- la **vérification** qu'un quadruplet $(x,\ y,\ z,\ t)$ forme effectivement une analogie
- la **résolution**, ou l'identification d'un élément t manquant à un triplet $(x,\ y,\ z)$ pour former une relation de proportion

Le triplet sur lequel est appliquée l'opération de résolution est appelé une équation analogique, notée $[\, x \,:\, y \,::\, z \,:?\,]$.

Stroppa et Yvon [Stroppa et Yvon, 2007] ont proposé, suivant les définitions introduites dans [Stroppa et Yvon, 2004], un cadre formel général permettant de calculer des proportions analogiques sur les mots (séquences), mais aussi sur les ensembles, les structures de traits et les arbres. Nous proposons dans les sections suivantes une description des algorithmes associés sur le domaine des séquences. Nous présentons ensuite brièvement les implications des formalisations sur les autres structures algébriques, ce qui nous permet d'exposer les arguments qui fondent notre choix d'utiliser les proportions

entre séquences pour le problème de transfert de langage. Enfin, nous décrivons en section 4.4 les optimisations et heuristiques proposées dans la littérature pour limiter le coût d'exécution des algorithmes sur les séquences.

4.2 Conventions de notation

Dans la suite de ce chapitre et des suivants, nous présentons différentes implémentations et optimisations des opérations de vérification et de résolution analogique, mais aussi de transfert au sens plus large en s'appuyant sur les proportions analogiques. Nous y employons certaines primitives dont nous précisons ici la signification :

`append(l, e)` : ajoute de l'élément `e` à la fin de la liste `l`.
`head(l)` : retourne l'élément en tête de la liste `l`.
`tail(l)` : retourne l'élément en queue de la liste `l`.
`n.children()` : retourne la liste des enfants du nœud `n` d'un arbre.
`<a, b, c>` : cette notation désigne un tuple contenant les attributs `a`, `b`, et `c`.

Nous étudions également la complexité algorithmique de ces implémentations par comparaison asymptotique à l'aide des notations de Landau. Nous utilisons en particulier la notation $\mathrm{O}(f(n))$, spécifiant que la durée d'exécution de l'algorithme en fonction de n est bornée asymptotiquement par la fonction f à un facteur près, ainsi que la notation $\Theta(f(n))$, spécifiant que la durée d'exécution de l'algorithme en fonction de n est soumise asymptotiquement à la fonction f à un facteur près. Cette dernière notation est plus précise car contrairement à la notation O avec laquelle la fonction n'est qu'une borne supérieure, le Θ ajoute l'information que la fonction f encadre le comportement de l'algorithme.

Lorsque l'expression de la comparaison asymptotique du temps d'exécution d'un algorithme n'est pas triviale, nous employons également la notation $\omega(f)$ qui donne en f une minoration asymptotique de la durée d'exécution à un facteur près. L'étude simultanée de $\mathrm{O}(f)$ et de $\omega(g)$ pour un algorithme permet d'affiner un peu l'estimation de sa complexité temporelle grâce à l'encadrement des fonctions f et g.

4.3 Définitions et algorithmes

Cette section présente les notions fondamentales nécessaires pour définir la validité d'une proportion analogique, et par extension la validité d'une solution à une équation analogique. Les algorithmes présentés dans cette section reprennent l'approche proposée initialement dans [Stroppa et Yvon, 2004], puis reprise dans [Stroppa, 2005] ainsi que dans des travaux plus récents sur lesquels nous revenons par la suite.

Vérification de proportionnalité

Une **proportion analogique**[*] entre quatre objets quelconques est définie formellement comme suit :

$$[x : y :: z : t] \Leftrightarrow (x, y) = (z, t) \vee (x, z) = (y, t)$$

Si les objets sont atomiques, alors il n'existe donc que deux cas possibles dans lesquels ils forment des proportions valides : $[a : a :: b : b]$ et $[a : b :: a : b]$. On appelle ces deux configurations les **proportions analogies canoniques**[*].

En revanche, lorsqu'il s'agit d'une structure complexe et que la décomposition est possible, alors la relation d'analogie s'applique récursivement aux propriétés atomiques respectives des objets. Dans

	a	d	v	e	n	a	i	t
	a	d	v	i	e	n	t	
c	o	n	t	e	n	a	i	t
c	o	n	t	i	e	n	t	

EXEMPLE 4.1 – Alignement analogique des caractères en alternances pour la proportion $[advenait : advient :: contenait : contient]$.

notre cas, nous nous intéresserons essentiellement aux séquences, grâce auxquelles on peut représenter des manifestations linguistiques sous forme de suites de mots, de caractères, ou encore de phonèmes. On appelle **atome**[*] l'unité considérée comme insécable pour le calcul de proportions analogiques. Nous écartons la représentation sous forme de phonèmes, car non adéquate aux données dont nous disposons, et par ailleurs non souhaitable à ce point comme précisé dans la section 2.2.

Dans le cas du raisonnement analogique sur les séquences, étant donné un quadruplet (X, Y, Z, T) il s'agit d'identifier un alignement des symboles de X, Y, Z et T tel que chaque quadruplet de symboles alignés (X_n, Y_n, Z_n, T_n) soit une proportion analogique valide. L'exemple 4.1 présente un alignement analogique entre quatre séquences de caractères.

Formellement, nous définissons un **alignement analogique**[*] $\mathcal{A}$ de quatre séquences comme suit :

$$\mathcal{A}(X,Y,Z,T) = \left(\mathcal{E}^X, \mathcal{E}^Y, \mathcal{E}^Z, \mathcal{E}^T\right), \forall i \leq \lambda \left[\mathcal{E}_i^X : \mathcal{E}_i^Y :: \mathcal{E}_i^Z : \mathcal{E}_i^T\right]$$

où la **taille**[*] de l'alignement est désignée par $\lambda \geq max\left(|X|, |Y|, |Z|, |T|\right)$, et $\mathcal{E}^S$ est l'*extension* à longueur λ de la séquence S. Les extensions à longueur n d'une séquence S ($n \geq |S|$) sont toutes les séquences de longueur n telles que le retrait des symboles vides ϵ donne S à nouveau.

$$E_n^S = \left\{\mathcal{E}^S | \mathcal{E}^S \setminus \epsilon = S \wedge \left|\mathcal{E}^S\right| = n\right\}$$

L'exemple d'alignement donné précédemment est de taille $\lambda = 9$. En pratique, il n'est pas intéressant de considérer des alignements analogiques de tailles supérieures à $|Y| + |Z|$ car ils incluent nécessairement au moins un quadruplet de symboles vides.

L'algorithme naïf de vérification de proportionnalité sur les séquences repose sur la notion de produit de mélange. Le **produit de mélange**[*] de deux séquences, noté $s_1 \bullet s_2$, est l'ensemble des séquences composées des symboles de s_1 et de s_2 en respectant leur ordre initial.

$$s_1 \bullet s_2 = \{m | \exists I \subseteq I_m, m(I) = s_1 \wedge m(I_m \setminus I) = s_2\}$$

I_m étant l'ensemble des indices des symboles de la séquence m, et $m(I)$ la sous-séquence de m constituée des symboles aux indices ordonnés de I. Un exemple de produit de mélange exhaustif est donné ci-dessous.

$$ab \bullet xy = \{abxy, axby, axyb, xaby, xayb, xyab\}$$

L'algorithme 4.1 produit l'ensemble des mélanges de deux séquences. Notons que le nombre de mélanges possibles croît exponentiellement en fonction des longueurs des séquences. Chaque mélange correspond en effet à l'un des plus courts chemins dans une grille $|s_1| \times |s_2|$:

$$|s_1 \bullet s_2| = \frac{(|s_1| + |s_2|)!}{|s_1|!|s_2|!}$$

Algorithme 4.1 : Produit de mélange de deux séquences

Entrées : Deux séquences s_1 et s_2
Sorties : L'ensemble $s_1 \bullet s_2$

si $|s_1| = 0$ **alors**
  retourner $\{s_2\}$
sinon si $|s_2| = 0$ **alors**
  retourner $\{s_1\}$
fin
$M \leftarrow \varnothing$
pour tous les $m \in tail(s_1) \bullet s_2$ **faire**
  $M \leftarrow M \cup \{append(head(s_1), m)\}$
fin
pour tous les $m \in s_1 \bullet tail(s_2)$ **faire**
  $M \leftarrow M \cup \{append(head(s_2), m)\}$
fin
retourner M

Un quadruplet de séquences (x, y, z, t) est une proportion analogique si et seulement si les symboles de x et de t apparaissent dans y et z dans le même ordre. Autrement dit, il existe un mélange de x et t et un mélange de y et z identiques :

$$[x : y :: z : t] \Leftrightarrow x \bullet t \cap y \bullet z \neq \varnothing$$

L'algorithme 4.2 permet de vérifier la proportionnalité d'un quadruplet de séquences en suivant directement cette définition. La complexité algorithmique exponentielle de l'opération de mélange et le produit cartésien nécessaire au calcul de l'intersection (ligne 3) rendent cet algorithme trop coûteux dès que l'on dépasse la taille d'exemples jouets, voir tableau 4.2 plus loin pour les analyses de complexité. Nous discutons des optimisations qu'il est possible d'y appporter dans la section 4.4.

Algorithme 4.2 : Vérification de proportion

Entrées : Un quadruplet de séquences x, y, z, t
Sorties : Un booléen indiquant si [x : y : : z : t]

1 $M_{xt} \leftarrow x \bullet t$
2 $M_{yz} \leftarrow y \bullet z$
3 $inter \leftarrow M_{xt} \cap M_{yz}$
4 **retourner** $inter \neq \varnothing$

Résolution d'équation analogique

La résolution d'une équation analogique s'appuie sur les mêmes principes que l'opération de vérification. Les solutions S d'une équation analogique $[x : y : z : ?]$ sont trivialement l'ensemble des t tels que le quadruplet (x, y, z, t) forme une proportion analogique valide.

$$S = \{t : [x : y :: z : t]\}$$

On peut alors reprendre la définition précédente de la proportion analogique à l'aide des mélanges afin d'obtenir une caractérisation de t par rapport à x, y et z :

$$\begin{aligned}[x : y :: z : t] &\Leftrightarrow x \bullet t \cap y \bullet z \neq \varnothing \\ &\Leftrightarrow \exists s \in x \bullet t \ : s \in y \bullet z \\ &\Leftrightarrow t \in \{s \setminus x : s \in (y \bullet z)\}\end{aligned}$$

Où $s_1 \setminus s_2$ désigne l'ensemble des sous-séquences obtenues à partir de s_1 par extraction des symboles de s_2 en respectant l'ordre, voir algorithme 4.3. Par exemple, $abcb \setminus ab = \{cb, bc\}$ et $abcb \setminus ca = \varnothing$.

Algorithme 4.3 : Complémentaire de séquences

Entrées : Deux séquences s_1 et s_2
Sorties : L'ensemble $s_1 \setminus s_2$

si $|s_2| = 0$ **alors**
  retourner $\{s_1\}$
sinon si $|s_1| = 0$ **alors**
  retourner $\varnothing$
fin
$C \leftarrow \varnothing$
si $head(s_1) = head(s_2)$ **alors**
  pour tous les $c \in tail(s_1) \setminus tail(s_2)$ **faire**
    $C \leftarrow C \cup \{c\}$
  fin
fin
pour tous les $c \in tail(s_1) \setminus s_2$ **faire**
  $C \leftarrow C \cup \{append(head(s_1), c)\}$
fin
retourner C

Suivant cette définition, on obtient l'algorithme 4.4 pour résoudre une équation analogique. Il s'appuie, comme l'algorithme de vérification, sur le calcul de mélanges. Cela lui donne également une complexité exponentielle en fonction de la longueur des séquences. Notons en outre que l'algorithme de complémentaire entre séquences, bien qu'il soit linéaire lorsque s_1 et s_2 n'ont pas ou peu de symboles en commun, est en réalité exponentiel également.

Algorithme 4.4 : Résolution d'équation

Entrées : Un triplet de séquences x, y, z
Sorties : La solution de [x : y : : z : ?]

$M_{yz} \leftarrow y \bullet z$
$S \leftarrow \varnothing$
pour tous les $m \in M_{yz}$ **faire**
  $S \leftarrow S \cup m \setminus x$
fin
retourner S

La résolution d'une équation analogique sur les séquences peut donner zéro, une ou plusieurs solutions. Les équations ayant une unique solution sont les équations analogiques ne mettant en jeu qu'un seul symbole pour chacune des séquences, telles que $[a : b :: a : ?]$ ou $[aa : aa :: a : ?]$. Toute autre équation analogique a zéro ou de multiples solutions. Une équation n'a aucune solution si, quelle que soit la séquence candidate t, il n'existe aucun alignement des symboles de x, y, z et t tel que tous

	a	d	v		e	n	a	i	t
	a	d	v	i		e	n	t	
c	o	n		t	e	n	a	i	t
c	**o**	**n**		**i**	**t**		**e**	**n**	**t**

EXEMPLE 4.2 – L'un des alignements duaux de celui présenté dans l'exemple 4.1.

solution	**occurrences (alignements)**
contient	11 410
conitent	4 780
cointent	3 040
conteint	2 471
ciontent	1 500
contietn	875

TABLEAU 4.1 – Les six solutions les plus occurrentes lors de la résolution de l'équation analogique $[advenait : advient :: contenait : ?]$.

les quadruplets de symboles alignés forment des proportions analogiques. Dans tous les autres cas, il existe donc une séquence t et un alignement associé mettant un jeu plus d'un symbole. Les tableaux ci-dessous montrent deux alignements pour l'équation analogique minimale non triviale $[a : ab : a : ?]$ ainsi que leurs solutions associées.

a	
a	b
a	
a	**b**

a		
a	b	
		a
	b	**a**

Ces alignements et solutions associées sont duaux car ils peuvent être obtenus l'un à partir de l'autre, par inversion des atomes alignés de part et d'autre de la frontière d'une alternance. Cette dualité est généralisable à toute équation analogique non triviale. En particulier, reprenant l'exemple donné plus haut, l'équation $[advenait : advient :: contenait : ?]$, admet aussi un autre alignement productif tel qu'illustré dans l'exemple 4.2.

Au total, 33 590 alignements permettent de produire une solution pour l'équation analogique $[advenait : advient :: contenait : ?]$, donnant 126 solutions distinctes. Cette multiplicité pose le problème de la sélection d'une réponse appropriée parmi l'ensemble des solutions. On peut alors s'interroger sur les propriétés effectives des solutions "naturelles" que l'on aimerait sélectionner.

Puisqu'il y a un nombre bien plus faible de solutions distinctes que d'alignements les produisant, on peut s'intéresser par exemple au nombre d'occurrences de chacune. Le tableau 4.1 donne les six solutions les plus occurrentes pour l'équation analogique étudiée. On constate qu'il y a un net avantage en faveur de la solution "naturelle" attendue. Il semble raisonnable d'adopter par induction l'hypothèse générale que la solution attendue à une équation analogique est la plus fréquente parmi les nombreuses produites par cet algorithme. C'est une hypothèse fréquemment admise dans la littérature [Stroppa et Yvon, 2006, Langlais *et al.*, 2009]. Évaluer sa pertinence est cependant délicat. En effet, elle repose fortement sur la notion de "solution attendue", qui n'est pas clairement définie. Plus précisément, cette notion résiste à la définition formelle du fait de sa dépendance à la langue naturelle. Une solution erronée dont on comprend le raisonnement sous-jacent est-elle réellement inattendue ?

algorithme	**complexité**	**approximation**
mélange de séquences	$\Theta\left(\frac{(\lvert s_1\rvert+\lvert s_2\rvert)!}{\lvert s_1\rvert!\lvert s_2\rvert!}\right)$	$\Theta\left(\frac{(2\lvert s\rvert)!}{\lvert s\rvert!^2}\right)$
complémentaire	$\mathrm{O}\left(2^{\lvert s_1\rvert-\lvert s_2\rvert}\right)$ [1]	$\omega\left(\lvert s_1\rvert\right)$
vérification par mélange	$\mathrm{O}\left(\frac{(\lvert s_1\rvert+\lvert s_4\rvert)!}{\lvert s_1\rvert!\lvert s_4\rvert!}\times\frac{(\lvert s_2\rvert+\lvert s_3\rvert)!}{\lvert s_2\rvert!\lvert s_3\rvert!}\right)$	$\mathrm{O}\left(\left(\frac{(2\lvert s\rvert)!}{\lvert s\rvert!^2}\right)^2\right)$
résolution par mélange	$\mathrm{O}\left(\frac{(\lvert y\rvert+\lvert z\rvert)!}{\lvert y\rvert!\lvert z\rvert!}\times 2^{\lvert y\rvert+\lvert z\rvert-\lvert x\rvert}\right)$	$\omega\left(\frac{(2\lvert s\rvert)!}{\lvert s\rvert!^2}\times\lvert s\rvert\right)$

TABLEAU 4.2 – Complexités des opérations analogiques sur les séquences par mélange

On constate cependant que les seules équations analogiques incorrectement résolues avec ce critère sont celles auxquelles des enfants sont également susceptibles d'échouer, ou autrement dit, celles nécessitant une information extérieure au triplet fourni. Par exemple :

- $[\textit{élire} : \textit{éligible} :: \textit{écrire} : ?]$
- $[\textit{coudre} : \textit{cousu} :: \textit{moudre} : ?]$
- $[\textit{informatique} : \textit{information} :: \textit{physique} : ?]$

Nous supposons donc provisoirement que la solution la plus pertinente d'une équation analogique est la plus fréquemment générée. Cette hypothèse pose néanmoins un problème pratique : l'énumération des solutions analogiques est coûteuse. Les complexités algorithmiques respectives des algorithmes présentés plus haut sont données dans le tableau 4.2.

Nous discutons en section 4.4 des heuristiques et optimisations apportées à ces algorithmes dans la littérature, ainsi que de leurs implications sur l'hypothèse posée pour la réponse la plus pertinente.

Au delà des proportions entre séquences

Le raisonnement analogique formel peut être appliqué à plusieurs types de structures algébriques. Un ensemble de définitions pour l'analogie formelle est proposé dans [Stroppa, 2005, Stroppa et Yvon, 2007] pour les séquences, mais aussi pour les ensembles, les multiensembles ou sacs, les structures de traits et les arbres.

Les opérations analogiques sur les ensembles, multiensembles et structures de traits simples (non récursives) ont des complexités algorithmiques linéaires en temps, et constantes ou linéaires en espace, ce qui les rend beaucoup plus efficaces que sur les séquences. Cependant, leur application au problème du transfert de langage requiert l'utilisation de méthodes d'analyse, rendant l'implémentation dépendante de la langue utilisée [2]. Cela contrevient à la contrainte de généricité que nous nous sommes fixé dans l'introduction. De plus, l'information réduite contenue dans les ensembles ou structures de traits manipulés, notamment à cause de la perte de l'ordre, risque d'augmenter le bruit dans la sortie en reconnaissant comme des proportions des quadruplets de séquences qui n'en sont pas.

Par ailleurs, les opérations analogiques sur des structures récursives comme les arbres ont une complexité algorithmique naïve exponentielle en fonction de la profondeur. En pratique cependant, des heuristiques permettent de conserver une complexité similaire à celle sur les séquences, grâce notamment à la linéarisation d'arbres [Stroppa et Yvon, 2007]. L'avantage de leur utilisation pour le transfert de langages vient de la restriction de l'espace des possibilités que permet la structuration des chaînes de caractères sous forme d'arbres. Les arbres donnent une segmentation et une hiérarchie

2. Il existe des travaux pour la conception d'analyseurs indépendants de la langue, mais leurs performances sont encore limitées et la discussion associée sort du cadre de notre travail.

Quel temps fait-il ?	:	Demande-lui le temps qu'il fait.
	: :	
Quel cours suit-elle ?	:	Demande-lui le cours qu'elle suit.

EXEMPLE 4.3 – Analogie et inversion de termes

qui limite la combinatoire des algorithmes. Ici aussi cependant, la construction des arbres syntaxiques requiert l'utilisation d'un analyseur et rend l'approche non générique. De plus, elle apporte également un risque d'erreurs ou d'absences de résultat dues à une analyse non appropriée. Notons cependant, comme il est souligné dans [Stroppa et Yvon, 2007], que l'analogie sur les arbres permet l'inversion de termes, qui ne peut être effectuée au niveau des séquences. Le quadruplet présenté dans l'exemple 4.3 n'est pas une proportion analogique valide sur les chaînes de caractères, mais elle est valide sur les arbres pour peu qu'on dispose d'une analyse adéquate de chacune des phrases.

Dans le cadre de la problématique du transfert de langages, nous avons préféré utiliser les modèles analogiques sur les séquences car elles conservent plus d'informations que ne le permettent les ensembles ou les structures de traits. D'autre part, les séquences sont plus génériques que les arbres et ne nécessitent aucun pré-traitement *ad hoc* avant application des algorithmes de raisonnement analogique.

4.4 Optimisations et heuristiques

Les algorithmes que nous avons présentés dans la section précédente ont des coûts computationnels élevés. Ce n'est pas satisfaisant si l'on veut les appliquer sur des séquences d'une longueur supérieure à une dizaine ou si ces opérations doivent être répétées un grand nombre de fois, comme ce sera le cas au chapitre suivant.

Concernant l'algorithme de résolution, nous avons vu que la sélection parmi les multiples solutions était faite à partir de leurs occurrences, ce qui implique la contrainte de les calculer exhaustivement. Une alternative est de calculer un échantillonnage aléatoire de ces solutions. Cette heuristique occasionne une probabilité d'erreur augmentant avec la diminution de la taille de l'échantillon.

D'autres heuristiques ont été proposées [Stroppa, 2005, Stroppa et Yvon, 2007] pour réduire le temps de calcul effectif nécessaire pour sélectionner une solution pertinente. Leur définition s'appuie sur la notion de factorisation de séquences. Une factorisation d'une séquence s est une suite de sous-séquences ou **facteurs*** $(s_1, ..., s_n)$ telle que $s_1 \cdot ... \cdot s_n = s$. On peut ainsi définir pour chaque alignement d'un quadruplet de séquences une factorisation correspondante pour chacune des séquences telles que chaque quadruplet de facteurs alignés, ou cofacteur, corresponde à l'une des deux proportions analogiques canoniques. Formellement, nous définissons la factorisation minimale associée à un alignement analogique de quatre séquences comme suit :

$$\mathcal{F}\left(\mathcal{A}\left(x,y,z,t\right)\right) = \left(\mathcal{F}^x, \mathcal{F}^y, \mathcal{F}^z, \mathcal{F}^t\right), \forall i \leq \theta \left[\mathcal{F}_i^x : \mathcal{F}_i^y :: \mathcal{F}_i^z : \mathcal{F}_i^t\right]$$

où la taille de la factorisation, appelée **degré*** de la proportion est $\theta = |\mathcal{F}^x| = |\mathcal{F}^y| = |\mathcal{F}^z| = \left|\mathcal{F}^t\right|$. L'exemple 4.4 présente deux alignements différents, de tailles 13 et 9 conduisant à une même factorisation de degré 2.

Stroppa et Yvon [Stroppa et Yvon, 2007] proposent l'utilisation d'algorithmes tabulaires pour la vérification et la résolution d'analogies. L'algorithme de résolution remplit une matrice tridimensionnelle correspondant à l'automate du langage des solutions analogiques. Il s'agit d'un automate à états finis non déterministe dont les états sont les cellules de la matrice, et les transitions, les liens établis par l'algorithme, voir [Stroppa et Yvon, 2006, Stroppa et Yvon, 2007]. Son état initial est la cellule

			1				2					
			a	d		v	e	n	a	i		t
			a	d		v	i	e		n	t	
c	o	n			t		e	n	a	i		t
c	o	n			t		i	e		n	t	
1	2	3	4	5	6	7	8	9	10	11	12	13

	1				2				
	a	d	v	e	n	a	i	t	
	a	d	v	i	e	n	t		
c	o	n	t	e	n	a	i	t	
c	o	n	t	i	e	n	t		
1	2	3	4	5	6	7	8	9	

EXEMPLE 4.4 – Deux alignements d'une équation analogique donnant la même factorisation, donc la même solution.

			1				2				3	
			a	d		v	e	n	a	i		t
			a	d		v	i	e		n		t
c	o	n			t		e	n	a	i	t	
c	o	n			t		i	e		n	t	
1	2	3	4	5	6	7	8	9	10	11	12	13

EXEMPLE 4.5 – Un alignement de degré différent pour une solution identique

$(0, 0, 0)$, et son état final la cellule $(|x|, |y|, |t|)$. Une fois l'automate constitué, l'énumération des séquences qu'il reconnaît donne l'ensemble des solutions analogiques de l'équation.

L'algorithme 4.5 correspond à la première étape de cette résolution : la constitution de l'automate. La complexité théorique de cet algorithme tabulaire complet est similaire à celle de la méthode par mélange et complément, car l'automate compte autant de chemins différents qu'il existe d'alignements des trois séquences de l'équation. Un terme en O $(|x| \times |y| \times |z|)$ lié au remplissage de la matrice est d'ailleurs ajouté à la complexité temporelle et spatiale. Cependant, la méthode naïve génère également des mélanges de y et z qui n'ont pas de complément par rapport à x, ce que ne fait pas l'algorithme tabulaire. Ce dernier parcourt ainsi en pratique une part réduite de l'espace de recherche. En outre, l'avantage principal de la méthode tabulaire est la possibilité de n'explorer qu'une partie de l'ensemble des solutions. Bien que cette heuristique était déjà applicable avec la méthode naïve par échantillonnage des mélanges, la méthode tabulaire pour sa part garantit de trouver une solution, s'il en existe une, quel que soit l'échantillonnage. De plus, il est possible de guider la recherche par des pondérations appliquées aux transitions de l'automate, de manière à explorer en priorité les chemins correspondants aux solutions présentant des propriétés spécifiques. Ces pondérations peuvent s'appuyer par exemple sur la taille, le degré ou le ϵ-order des solutions [3]. Nous avons déjà introduit les notions de taille et de degré de proportions analogiques. Le ϵ-order d'une proportion analogique, introduit dans [Stroppa, 2005], est le nombre maximum de facteurs vides consécutifs dans sa factorisation minimale associée. La notion de degré de l'analogie correspond particulièrement bien au nombre d'occurrences des solutions, sur lequel se base l'heuristique de sélection naïve. En effet, comme le montre l'exemple 4.4, les multiples alignements correspondant à une même solution correspondent également à une même factorisation. Les ajouts de symboles vides augmentant la taille de l'alignement n'influencent pas l'alternance des facteurs, excepté dans les cas exceptionnels où la proportion contient un quadruplet de symboles tous identiques. Ainsi, l'alignement de l'équation montré dans l'exemple 4.5 conduit à la même solution qu'obtenue dans l'exemple 4.4, bien qu'il diffère par son degré.

Contrairement au nombre total d'alignements ou chemins du graphe donnant la même solution,

3. La taille, le degré ou le ϵ-order sont notamment calculés sur des solutions partielles pour les transitions conduisant à des états non finaux du graphe, *i.e.* les liens vers des cellules (i, j, k) de la matrice, telles que $i + j + k < [x| + |y| + |z|$.

Algorithme 4.5 : Résolution d'équation analogique (tabulaire)

Entrées : Un triplet de séquences x, y, z
Sorties : Un énumérateur vers l'ensemble des solutions de [x : y : : z : ?]

pour tous les $(i, j, k) \in \{-1, 0\}^3$ **faire**
  $s(i, j, k) \leftarrow \varnothing$
fin
pour $i \leftarrow 0, |x|$ **faire**
  pour $j \leftarrow 0, |y|$ **faire**
    pour $k \leftarrow 0, |z|$ **faire**
      si $i = j = k = 0$ **alors**
        $s(i, j, k) \leftarrow \{\}$
      sinon
        $s(i, j, k) \leftarrow s(i-1, j-1, k)$ if $x(i) = y(j)$
        $\cup \; s(i-1, j, k-1)$ if $x(i) = z(k)$
        $\cup \; s(i, j-1, k).y(j)$
        $\cup \; s(i, j, k-1).z(k)$
      fin
    fin
  fin
fin
retourner $s(i, j, k)$

le degré de la proportion peut-être calculé à l'aide seulement du chemin courant dans la résolution. En effet, le degré augmente d'une unité pour chaque passage d'un type d'alternance à l'autre. Nous proposons d'appeler les alternances de type $[a : a :: b : b]$ **alternances *plates****, et les alternances de type $[a : b :: a : b]$ **alternances *croisées****. Dans l'exemple précédent, les changements de types d'alternances sont entre les alignements unitaires 7 et 8, d'alternance plate à croisée, et entre les 11 et 12, de croisée à plate. On peut donc calculer et mettre à jour dans chaque cellule de la matrice le degré minimal et le chemin associé y conduisant. Nous proposons donc une version adaptée du remplissage de la matrice tridimentionnelle permettant, une fois terminé, de retrouver la solution de degré minimal en temps linéaire. Rappelons que la complexité de la recherche de la solution de degré minimal après constitution du graphe des solutions est celle de l'algorithme du plus court chemin entre l'entrée $(0, 0, 0)$ et la sortie $(|x|, |y|, |z|)$ du graphe. L'algorithme de Dijkstra [Dijkstra, 1971, p. 67] requiert $\mathrm{O}\left(A + N \times log\left(N\right)\right)$ opérations pour un graphe comportant N nœuds et A arrêtes. Dans notre cas, on a $N = |x| \times |y| \times |z|$ et $A \approx 2 * N$ pour les grandes tailles de N. Ce qui donne une complexité générale[4] en $\mathrm{O}\left(N^3\left(6 + log\left(N^3\right)\right)\right)$. En comparaison, la complexité en mémorisant le plus court chemin de manière incrémentale donne $\mathrm{O}\left(4 + N^3\right)$.

D'autre part, on peut noter également que, par suite de la remarque précédente, la correspondance entre la solution la plus occurrente et la solution associée au degré minimal se transmet aussi à la solution de taille minimale. En effet, l'alignement le plus compact (*i.e.* de taille minimale) correspond nécessairement au degré minimal de la proportion. Réciproquement pour une solution de degré minimal, tous les alignements de taille minimale la produisent. Il en découle que l'optimisation par mémorisation du degré dans les cellules de la matrice peut également être appliquée pour la taille des alignements analogiques. La différence est que contrairement au degré, les tailles des alignements

4. La complexité est ici donnée tenant compte des coefficients constants pour faire apparaître les nuances dans les algorithmes.

compressés ne suivent pas la courbe des occurrences. Leur utilisation ne permet donc pas d'ordonner l'ensemble des solutions de la même manière que pour le degré, seule la meilleure solution conserve la valeur minimale de taille comme de degré. Cependant, l'avantage de l'utilisation de la taille est qu'elle est triviale à calculer dans chaque cellule, valant toujours un de plus que la taille dans la cellule parente. Il permet donc de conserver une complexité proche de l'algorithme de remplissage seul : $\mathrm{O}\left(4 \times N^3\right)$

Quant à l'algorithme de vérification, il fonctionne en remplissant une matrice quadridimensionnelle par des booléens représentant si l'un des chemins conduisant vers cette cellule est valide. Il est détaillé dans l'algorithme 4.6.

Algorithme	Temps
Résolution tabulaire (algorithme 4.5)	$O\left(s^4\right)$
Résolution tabulaire optimisée (algorithme 4.5)	$O\left(s^3\right)$
Vérification tabulaire (algorithme 4.7)	$O\left(s^3\right)$

TABLEAU 4.3 – Complexités pour vérification et résolution d'analogies

Algorithme 4.6 : Vérification de proportion (tabulaire)

Entrées : Un quadruplet de séquences x, y, z, t
Sorties : Un booléen indiquant si [x : y : : z : t]

pour tous les $(i, j, k, l) : i = -1 \vee j = -1 \vee k = -1 \vee l = -1$ **faire**
  $a[i][j][k][l] \leftarrow$ **faux**
fin
pour $i = 0 \rightarrow |x|$ **faire**
  pour $j = 0 \rightarrow |y|$ **faire**
    pour $k = 0 \rightarrow |z|$ **faire**
      pour $l = 0 \rightarrow |t|$ **faire**
        si $i = j = k = l = 0$ **alors**
          $a[i][j][k][l] \leftarrow$ **vrai**
        sinon
          si $a[i-1][j-1][k][l] \wedge x[i] = y[j]$
          $\vee\, a[i-1][j][k-1][l] \wedge x[i] = z[k]$
          $\vee\, a[i][j-1][k][l-1] \wedge t[l] = y[j]$
          $\vee\, a[i][j][k-1][l-1] \wedge t[l] = z[k]$ **alors**
            $a[i][j][k][l] \leftarrow$ **vrai**
          sinon
            $a[i][j][k][l] \leftarrow$ **faux**
          fin
        fin
      fin
    fin
  fin
fin
retourner $a[|x|][|y|][|z|][|t|]$

Cet algorithme peut être optimisé pour une complexité cubique en appliquant une résolution contrainte. Autrement dit, la matrice n'est pas remplie avec tous les chemins possibles mais on lit

en entrée la quatrième de proportion pour dérouler les seuls chemins qui lui correspondent. L'algorithme 4.7 détaille cette optimisation.

4.5 Conclusion

Dans ce chapitre, nous avons présenté les fondements théoriques et pratiques du raisonnement par analogie formelle, en particulier lorsqu'il est appliqué sur des séquences. Les deux méthodes nécessaires pour le calcul des analogies sont l'opération de vérification d'une proportion, et l'opération de résolution d'une équation analogique en vue d'obtenir l'ensemble des solutions correspondantes. Chacune de ces méthodes est très coûteuse en temps d'exécution, de par la multiplicité des solutions possibles. Nous avons vu que des propositions d'optimisations et d'heuristiques pour ces dernières, soulageant la charge d'exécution, et dont les complexités algorithmiques sont données dans le tableau 4.3 Dans la suite, nous n'utiliserons sauf mention contraire que l'algorithme tabulaire de résolution ainsi que la mémorisation des tokens.

Algorithme 4.7 : Vérification de proportion optimisée (tabulaire)

Entrées : Un quadruplet de séquences x, y, z, t
Sorties : Un booléen indiquant si [x : y : : z : t]

si $|x| + |t| \neq |y| + |z|$ **alors**
  retourner faux
fin
pour $i = 0 \rightarrow |x|$ **faire**
  pour $j = 0 \rightarrow |y|$ **faire**
    pour $k = 0 \rightarrow |z|$ **faire**
      si $i = j = k = 0$ **alors**
        $a[i][j][k] \leftarrow 0$
      sinon
        $l \leftarrow$ **nil**
        si $x[i] = y[j]$ **alors**
          $l \leftarrow a[i-1][j-1][k]$
        fin
        si $l =$ **nil** $\wedge\, x[i] = z[k]$ **alors**
          $l \leftarrow a[i-1][j][k-1]$
        fin
        $l_{tmp} \leftarrow a[i][j-1][k]$
        si $l =$ **nil** $\wedge\, l_{tmp} \neq$ **nil** $\wedge\, l_{tmp} < |t| \wedge t[l_{tmp}+1] = y[j]$ **alors**
          $l = l_{tmp} + 1$
        fin
        $l_{tmp} \leftarrow a[i][j][k-1]$
        si $l =$ **nil** $\wedge\, l_{tmp} \neq$ **nil** $\wedge\, l_{tmp} < |t| \wedge t[l_{tmp}+1] = z[k]$ **alors**
          $l = l_{tmp} + 1$
        fin
        $a[i][j][k] \leftarrow l$
      fin
    fin
  fin
fin
si $a[|x|][|y|][|z|] = |t|$ **alors**
  retourner vrai
sinon
  retourner faux
fin

Chapitre 5

Analogie pour le transfert de langages

Sommaire

Imprime 2 copies du fichier liste.txt : `lp -n 2 liste.txt`
::
Imprime 3 copies du fichier doc.pdf : `?`

EXEMPLE 5.1 – Équation analogique pour un transfert direct

5.1 Introduction

Nous avons introduit dans le chapitre précédent le cadre formel pour la manipulation d'analogies sur les séquences. Le présent chapitre a pour objectif de présenter l'application de ces méthodes pour le problème du transfert de langage. En particulier, il s'agit de déterminer comment seront utilisés les exemples de traduction disponibles dans la base de connaissances pour constituer des équations analogiques pertinentes.

Nous présentons d'abord une méthode directe de transfert en section 5.2, utilisant une simple équation pour transférer les éléments communs d'un langage à l'autre. Dans la section 5.3, nous décrivons des approches plus élaborées, fondées sur la découverte d'analogies internes à chaque langage. Ces approches appelées indirectes souffrent d'un coût algorithmique élevé, et nous décrivons des méthodes utilisées dans la littérature pour réduire leur temps d'exécution. Nous proposons en section 5.5 une méthode pour enrichir la base de connaissances disponible à l'aide de l'analogie. Enfin, la section 5.7 est dédiée à la présentation de la mise en place expérimentale utilisant les approches présentées dans les sections précédentes ainsi qu'à l'analyse critique des résultats qu'elles ont permis d'obtenir.

5.2 Transfert analogique direct

La manière naïve d'envisager le transfert de langage par analogie consiste à former des triplets directement à partir des exemples présents dans la base. Prenons par exemple la requête "*Imprime 3 copies du fichier doc.pdf*", le principe est de rechercher une association requête-commande existante afin d'obtenir une équation analogique comme celle présentée dans l'exemple 5.1. On définit ainsi le **transfert direct*** comme la génération de l'équivalent en langue cible de l'énoncé source soumis à partir d'équations analogiques transverses, *i.e.* comprenant énoncés en langue source et leur association en langue cible.

On applique l'algorithme de résolution analogique pour chaque équation ainsi constituée. Pour celles qui ont des solutions, on ne considère que la solution de degré minimal. Dans l'exemple présenté, il s'agit donc de la commande "`lp -n 3 doc.pdf`", pour un degré de 4. On obtient donc un ensemble de solutions parmi lesquelles il faut déterminer la réponse finale à proposer. Avant d'envisager des critères de tri pour cet ensemble, tentons d'abord d'estimer le nombre de solutions distinctes auquel on peut s'attendre.

Rappelons qu'un quadruplet de séquences (x, y, z, t) est en proportion analogique si et seulement s'il existe un alignement dont chaque quadruplet atomique est de la forme $[a \ : \ a \ :: \ b \ : \ b]$ ou $[a \ : \ b \ :: \ a \ : \ b]$. Cela implique que tous les symboles de x sont contenus dans y ou dans z. Les symboles de x absents de z sont donc nécessairement présents dans y si la proportion est vérifiée. Par ailleurs, cette règle ne dépend pas de l'atome choisi : caractères ou tokens. Pour les situations considérées dans la suite, nous utilisons les tokens comme atomes de segmentation des séquences. Par corollaire de la règle énoncée, pour qu'une solution existe tous les tokens de x absents de z sont nécessairement présents dans y. En particulier, lorsque x et z sont systématiquement des phrases du langage source, et y et t des phrases du langage cible, il ne peut pas s'agir que de tokens qui peuvent être utilisés indifféremment dans les deux langages. Autement dit, un tel token représente pour les deux

langages la même unité de sens. Dans le cas particulier du transfert depuis la langue naturelle vers le langage formel, cela implique que tous les tokens de la requête z exclusifs à la langue naturelle [1] doivent être présents à la fois dans une requête de la base d'exemples. De plus, l'alignement des séquences contraint à conserver leur ordre d'apparition. Les seuls tokens qui échappent à ces contraintes sont les tokens communs aux deux langages que nous avons mentionnés. Ces tokens correspondent dans ce cas aux paramètres de la requête. Dans l'exemple précédent, il s'agit de "3" et "doc.pdf" qui pourront, et devront, apparaître dans la commande pour résoudre l'équation analogique et produire une solution correcte.

Il s'agit donc d'une forte limitation de l'approche directe puisqu'elle nécessite, pour produire une solution, d'identifier un exemple dans la base dont la requête soit tout à fait similaire à la requête soumise. Elle doit se conformer au même modèle que la requête soumise comme vu dans l'exemple 5.1, seuls les paramètres pouvant varier.

Les réponses que l'on peut attendre par cette approche directe ne sont pas uniques. En particulier, on peut obtenir des doublons dans les cas de figure suivants :

- la base contient plusieurs exemples dont la requête est construite sur le même modèle
- le modèle de requête est associé à plusieurs commandes différentes dans la base

Le premier cas ne donnera pas lieu à des réponses distinctes car les paramètres seront partout remplacés par ceux de la requête soumise. Le second cas produira des réponses distinctes, et cela conduit à une ambiguïté lorsque les différentes commandes ne sont pas synonymes. Dans ce dernier cas, plus d'informations sont nécessaires pour sélectionner la réponse attendue. Nous supposerons que la solution la plus occurrente est la plus probablement correcte car elle est plus populaire dans la base d'exemples. Cependant d'autres discriminants peuvent être utilisés tels que le degré de l'équation analogique mise en jeu, ou son succès au cours de précédentes interactions consignées dans une base d'apprentissage méta. La constitution d'une telle base permet l'utilisation de méthodes d'apprentissage artificiel afin d'identifier des critères des proportions analogiques permettant de prédire le succès de leur résolution et/ou la validité de la solution produite.

L'approche de transfert par analogie directe permet donc de déterminer la correspondance cible d'un énoncé en langage source, mais requiert de former grâce à la base une équation analogique respectant une somme stricte sur les symboles. De plus la méthode impose que tout token ou expression qui n'est pas directement transféré doit se trouver dans l'énoncé source extrait de la base d'exemples. En pratique, cette approche n'est pas utilisée pour la traduction d'une langue naturelle vers une autre langue naturelle à cause de ces limitations. En effet, les variations syntaxiques sont très nombreuses dans la langue naturelle non contrainte, et il n'est donc pas réaliste de supposer qu'une base pourra contenir suffisamment de variations pour traduire efficacement des phrases plus longues que quelques tokens. D'autre part, les mots communs entre deux langues naturelles sont rares. Il s'agit d'entités nommées, par exemple des noms de personnes ou de lieux, uniquement quand elles n'ont pas de traduction/translittération (exemples d'exceptions : Londres, Il Rodano, Estados Unidos). Les cognats [2] ne rentrent pas dans cette définition car ils ont souvent des différences morphologiques ou flexionnelles. Par exemple, les mots *nuit*, *night*, et *Nacht* sont cognats, mais le transfert de l'un vers l'autre

1. On entend par token d'une requête z exclusif à la langue naturelle un token qui n'est pas retrouvé dans la ou les commandes correspondant à z. Il s'agit en général de tous les mots de la langue (par opposition aux expressions numériques ou non lexicales), à l'exception de certains mots pouvant être repris comme noms de paramètre tels que "arbre", "liste" ou "data".

2. Des cognats sont des mots de langues distinctes qui sont proches voire identiques du fait de leur origine étymologique commune.

EXEMPLE 5.2 – Deux proportions analogiques en correspondance (tiré de [Lepage et Denoual, 2005])

I'd like to open these windows.	:	*Could you open a window ?*	: :	*I'd like to cash these traveler's checks.*	:	*Could you cash a traveler's check ?*
↕		↕		↕		↕
Est-ce que ces fenêtres, là, je peux les ouvrir ?	:	*Est-ce que vous pouvez m'ouvrir une fenêtre ?*	: :	*Ces chèques de voyage, là, je peux les échanger ?*	:	*Vous pouvez m'échanger un chèque de voyage ?*

n'est pas trivial malgré leur parenté. La rareté de ces transferts directs d'expressions entre langues naturelles rend cette approche analogique naïve proche d'un apprentissage par cœur. Cela l'exclut donc du paysage des approches réalisables à cause du nombre extrême de formulations possibles dans la langue naturelle.

D'autres approches ont été proposées pour la traduction de langues naturelles par analogie, permettant d'éviter ces limitations. Nous les présentons dans la section suivante et discutons également de leurs propriétés par rapport à l'approche directe.

5.3 Transfert analogique indirect

La relation de proportion analogique entre séquences est très stricte. Cela cause une rigidité du transfert de langages par analogie directe. Pour le transfert de la langue naturelle vers le langage formel, le modèle de la requête soumise doit déjà être connu pour pouvoir proposer une réponse.

Pour dépasser ce problème, une approche indirecte a été proposée dans [Lepage et Denoual, 2005]. Le **transfert analogique indirect*** consiste en la génération de l'équivalent en langue cible d'un énoncé en langue source soumis à partir d'équations analogiques homogènes, *i.e.* comprenant exclusivement des énoncés source ou exclusivement des énoncés cible. Cette méthode requiert pour cela la constitution simultanée d'une proportion analogique et d'une équation associée, comme nous le verrons dans ce qui suit. Nous présentons cette approche à partir des travaux de Lepage et Denoual, ainsi qu'une variante que nous proposons dans les prochaines sections.

5.3.1 Principe et algorithmes

Les approches pour la traduction automatique par analogie proposent d'effectuer le transfert analogique de manière implicite en travaillant uniquement sur la production d'énoncés au sein d'une même langue. Dans la suite, nous utilisons les notations suivantes : B est la base d'exemples, B_s et B_c sont respectivement ses projections sur le domaine source et sur le domaine cible, e_u^s est l'énoncé à traduire (considéré soumis par l'utilisateur), et e_u^c désigne une traduction proposée de cet énoncé. À partir d'un énoncé en langue source, on constitue une équation analogique d'énoncés source avec l'énoncé soumis par l'utilisateur, complété à l'aide de la base d'exemples. On résoud ensuite cette équation et on vérifie si l'énoncé cible solution est présent dans la base d'exemples. Si non, on cherche à le traduire récursivement. Si oui, alors on dispose donc d'un triplet d'énoncés source de la base. On forme ensuite une équation analogique à partir des trois énoncés cible correspondants dans la base. On retourne enfin les solutions de cette équation. L'exemple 5.2, extrait de [Lepage et Denoual, 2005], permet d'illustrer cette méthode. Elle est motivée notamment par l'hypothèse que les proportions analogiques de forme et de sens dans la langue source correspondent à des proportions analogiques de forme et de sens dans la langue cible.

Notons que l'ensemble des solutions calculées selon ce modèle est potentiellement grand. En effet, chacune des étapes de l'algorithme produit non pas une mais un ensemble de solutions :

- plusieurs proportions analogiques valides peuvent être formées dans le domaine source,
- à chaque énoncé source peuvent correspondre plusieurs énoncés cibles,
- chaque équation analogique formée dans le domaine cible peut produire plusieurs solutions.

Dans la suite, nous considérons que la première de ces trois sources de combinatoire est la plus importante. D'une part, la multiplicité des associations source-cible peut être contrôlée et on suppose qu'elle se situe toujours au dessous d'un seuil constant et négligeable devant la taille de la base d'exemples B (il est de fait borné en moyenne par $\frac{|B|}{|B_s|}$). D'autre part, seule une très faible part des solutions valides d'une équation analogique est réellement intéressante, et on dispose d'heuristiques pour les sélectionner (*cf.* section 4).

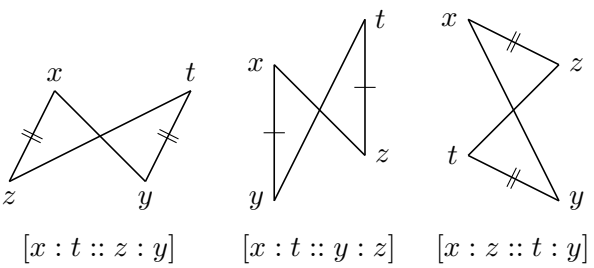
L'approche naïve de transfert par analogie suivant ce modèle est l'énumération exhaustive de toutes les proportions analogiques $[e_1^s : e_2^s :: e_3^s : e_u^s]$ avec $(e_1^s, e_2^s, e_3^s) \in {B_s}^3$. Elle implique donc $\mathrm{O}\left(|B_s|^3\right)$ opérations dont chacune a une complexité correspondant à la composition des deux dernières étapes, et qui ne dépend pas de la taille de la base [3].

Lepage et Denoual [Lepage et Denoual, 2005] proposent un algorithme de résolution récursif fondé sur la constitution dans le domaine source non pas de quadruplets à vérifier, mais de triplets de phrases afin de former des équations analogiques à résoudre. Nous reportons ci-dessous les étapes principales de leur approche. Les notations suivantes sont utilisées : B est la base d'exemples, B_s et B_c correspondent respectivement aux projections de la base sur la langue source et sur la langue cible, p^s est la phrase en langue source à traduire.

1. Énumérer tous les $(s_1, s_2) \in {B_s}^2$ et resoudre les équations analogiques associées de la forme $[s_1 : s_2 :: ? : p^s]$
2. Pour les solutions x^s de ces équations :
   - Si $x^s \in B_s$ alors constituer toutes les équations de la forme $[c_1 : c_2 :: x^c : ?]$ telles que $\{(s_1, c_1), (s_2, c_2), (x^s, x^c)\} \subset B$
   - Si $x^s \notin B_s$ alors traduire x^s récursivement avec le même algorithme puis ajouter les associations (x^s, x^c) obtenues à B
3. Retourner toutes les solutions y des équations cibles ainsi constituées (triées par nombre d'occurrences)

La complexité de cet algorithme dépend du nombre de phrases en langue source dans la base $|B_s|$ et du nombre d'associations source-cible $|B|$. Sans considérer la récursion et en supposant constant le temps de résolution analogique (par exemple pour une taille bornée des phrases de la base), cette complexité est dans le pire des cas en $\mathrm{O}\left(|B_s|^2 \times |B|\right)$. Le facteur $|B|$ concerne l'opération de recherche des phrases cibles associées à une phrase source dans la base. Cette opération nécessite évidemment bien moins que $|B|$ étapes en moyenne. Dans la suite, on supposera que le nombre de phrases cibles associées à une phrase source est borné par une constante $K \ll |B|$, ce qui permet d'ignorer son expression dans la formule de complexité globale.

3. Nous supposons également que la longueur des énoncés, dont dépend la complexité algorithmique de la résolution, est homogène sur la base d'exemples.



EXEMPLE 5.3 – Permutations non équivalentes de la proportion $[x : y :: z : t]$

En ce qui concerne l'algorithme complet, en supposant que la même paire (s_1, s_2) n'est utilisée qu'à un seul niveau de profondeur à la fois, la profondeur de récursion maximale est de $|B_s|^2$. La formule de complexité sans récursion devient alors le facteur de branchement [4] de la récursion ce qui donne dans le pire des cas $\mathrm{O}\left(|B_s|^2 \uparrow\uparrow 2\right)$. Le facteur de branchement est bien entendu décrémenté à chaque niveau de profondeur, mais le temps d'exécution reste trop élevé et nécessite l'emploi d'heuristiques. En pratique, Lepage et Denoual ont proposé un parcours des s_2 d'abord les plus similaires à p^s, puis des s_1 d'abord proches de s_2. La récursion est effectuée en un parcours en largeur d'abord, c'est-à-dire commençant par épuiser les premiers niveaux de récursion. Enfin, des limites ont été posées quant à la durée maximale d'exécution pour chaque récursion.

La section suivante présente différentes optimisations et heuristiques de l'état de l'art pour les algorithmes de parcours de la base que nous avons présentés précédemment.

5.3.2 Optimisations et heuristiques

L'approche indirecte pour le transfert de langage par analogie repose, nous l'avons vu, sur la constitution de proportions analogiques valides dans le domaine source. La méthode naïve comme la méthode récursive sont très coûteuses et ne peuvent passer à l'échelle. Afin de réduire la charge computationnelle, on cherche à effectuer une pré-sélection des énoncés source à considérer.

En premier lieu, la réduction de la charge computationnelle peut être effectuée en évitant toute redondance du processus de vérification ou de résolution. De nombreuses proportions apparemment distinctes sont en réalité équivalentes par "symétrie" analogique. Exploiter cette symétrie permet donc d'alléger la liste des triplets d'énoncés source par comparaison avec leur énumération naïve.

Les proportions analogiques possèdent deux propriétés fondamentales, établies en tant qu'axiomes : l'équivalence par échange des moyennes (éléments autour du symbole "::") et par inversion des ratios (éléments autour des ":"). Ces propriétés permettent de générer par composition un ensembles de huit formes représentatives de la même proportion [Lepage, 2004]. Elles sont listées dans la figure 5.1, et illustrées par des représentations géométriques subissant les mêmes transformations ramenées à des isométries du plan. L'exemple 5.3 donne un échantillon des 16 autres permutations possibles du quadruplet. Ces permutations ne peuvent pas être obtenues par composition des propriétés fondamentales que nous avons citées, ni dans leur version graphique par une composition de transformations homogènes du plan. Elles ne préservent pas les rapports entre distances, ni les proportions analogiques entre les objets.

Tenir compte de ces permutations équivalentes des proportions analogiques permet de réduire le nombre d'itérations nécessaires pour lister les proportions existantes dans le domaine source. Si l'on

4. Le symbole flèche utilisé correspond à la notation de Knuth [Knuth, 1976]. En particulier, on a $x \uparrow 2 = x \times x$, et $x \uparrow\uparrow 2 = x^x$.

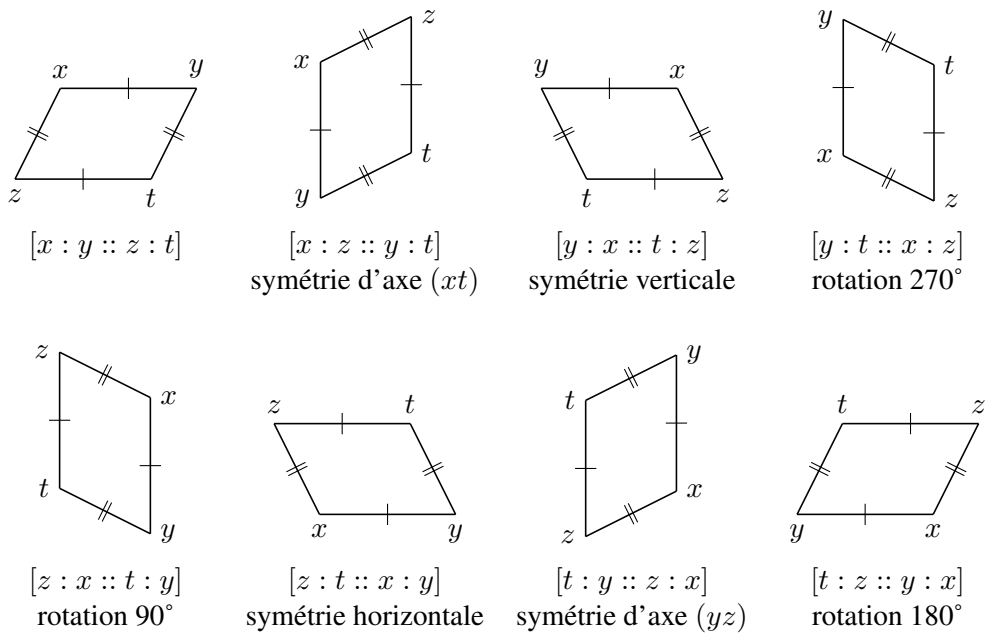


FIGURE 5.1 – Classe d'équivalence de proportions analogiques

fixe l'énoncé e_u^s à traduire en dernière position et un énoncé de la base e_i^s en première position, il ne reste qu'une proportion unique pour chaque paire (e_j^s, e_k^s), car $[e_i^s : e_j^s :: e_k^s : e_u^s] \equiv [e_i^s : e_k^s :: e_j^s : e_u^s]$. Il n'est donc nécessaire d'énumérer que les triplets (e_i^s, e_j^s, e_k^s) tels que $j < k$. Par ailleurs, les triplets où $i = j \vee i = k \vee j = k$ peuvent également être ignorés, car ils correspondent aux cas pour lesquels, respectivement $e_k^s = e_u^s \vee e_j^s = e_u^s \vee e_i^s = e_j^s = e_k^s = e_u^s$. Il s'agit des cas triviaux dans lesquels l'énoncé e_u^s est déjà présent dans la base. Il suffit donc de l'identifier au préalable en un temps au plus proportionnel à $|e_u^s|$ et de retourner l'ensemble des énoncés cibles associés dans la base d'exemples.

Ces optimisations directes ne concernent que les propriétés générales des proportions analogiques. Plusieurs travaux ont proposé des méthodes pour atteindre des coûts de calcul raisonnables en exploitant les propriétés spécifiques des proportions de séquences. [Somers *et al.*, 2009] rassemble et compare les approches développées jusqu'en 2009 et que nous présentons ci-dessous.

Stroppa et Yvon [Stroppa et Yvon, 2007] montrent que les relations de proportionnnalité analogique sont préservées par le morphisme de semigroupes. Cette propriété peut être utilisée pour optimiser la vérification et la résolution d'analogies grâce à des morphismes choisis pour simplifier ces calculs. L'un d'entre eux est le morphisme depuis l'ensemble des séquences muni de la concaténation $(S, \cdot)$ vers l'ensemble des comptages (entiers) muni de la somme $(\mathcal{N}, +)$. Cela implique en particulier que la proportion (additive) des comptages des symboles est nécessaire pour la proportion des séquences. Langlais et Yvon [Langlais et Yvon, 2008] proposent une structure d'arbre de comptage pour indexer l'ensemble de la base d'exemples selon les sommes de comptage de chaque séquence par rapport à chaque token possible. Un **arbre de comptage*** est une structure de données permettant d'accéder aux séquences qu'il indexe à l'aide de l'ensemble des occurrences des symboles qu'elles contiennent, appelé **vecteur de comptage***. Comme pour toute structure arborescente, cet accès est fait en temps logarithmique [5] en fonction de la taille de l'arbre en nombre de feuilles, c'est-à-dire dans notre cas le nombre de séquences indexées. Cette structure, dont la construction est présentée dans l'al-

5. Il s'agit ici du logarithme en base $|A|$ de la taille de l'arbre, A étant l'alphabet (ou lexique) utilisé.

gorithme 5.1, est utilisée pour retrouver à partir du vecteur de comptage d'une séquence ou d'une paire de séquences, l'ensemble des paires associées de la base totalisant le même comptage, et donc susceptibles de former des proportions analogiques. Aucune des paires existantes et non retournées n'aurait pu constituer une proportion analogique avec la ou les séquences en paramètre. L'algorithme 5.4 détaille la recherche dans un arbre de comptage. Il retourne un sous-ensemble de B à énumérer pour identifier des proportions analogiques dans le domaine source.

Algorithme 5.1 : Construction de l'arbre de comptage
La primitive *encode* permet de calculer le vecteur de comptage associé à son ou à ses paramètres.
La fonction *synchronise* correspond à l'algorithme 5.2
La fonction *insère* correspond à l'algorithme 5.3

Entrées : Un alphabet A et un ensemble I de séquences sur cet alphabet
Sorties : L'arbre de comptage T correspondant à l'ensemble I

```
1  T ← nil
2  pour chaque s ∈ I faire
3  |  counts ← encode(s)
4  |  ⟨current, parent, i⟩ ← synchronise(counts, T)
5  |  tant que i ≤ |A| faire
6  |  |  ⟨current, parent⟩ ← insère(counts, i, current, parent, A)
7  |  |  i ← i + 1
8  |  fin
9  |  current.forms ← current.forms ∪ {s}
10 fin
11 retourner T
```

La complexité algorithmique *dans le pire cas* en utilisant une optimisation par comptage est plus élevée que pour l'énumération exhaustive des triplets : $O\left(max_{c=|children(n)|:n\in T}\left(|A| \times c^2\right)\right)$
En effet, d'une part, une étape supplémentaire est nécessaire pour effectuer la sélection initiale du sous-ensemble de la base, et d'autre part, cette sélection ne garantit pas que le sous-ensemble est strictement plus petit que la base d'exemples elle-même. En pratique cependant, la durée réelle d'exécution est bien inférieure à celle pour l'énumération naïve, car la structure arborescente optimisée permet d'élaguer très rapidement de grands sous-espaces de recherche non productifs. Seulement dans le pire cas, où la base contient toutes les répartitions possibles (au dessous d'un certain seuil d'occurrences) des symboles dans les énoncés, la durée d'exécution pour l'énumération par comptage peut dépasser celle de l'énumération naïve, mais ce cas dégénéré ne représente que très peu de situations, qui de plus ne correspondent pas à des formulations naturelles puisqu'elles sont exhaustives (*cf.* la loi d'Estoup-Zipf sur la distribution des phénomènes linguistiques, mentionnée au chapitre 2).

5.4 Complémentarité des approches

Le transfert analogique direct présenté en section 5.2 ne permet pas d'obtenir des résultats satisfaisants dans le cas général. Lorsqu'il s'agit de remplacer un segment commun par un autre segment commun, les proportions croisées ont des solutions fortuites (segments "proportion" et "assertion" dans l'exemple 5.4.a). Cependant, dans le cas général où un segment commun aux deux langues est remplacé par un segment différent, le résultat produit est invalide (exemple 5.4.b).

La faible proportion de ces occurrences dans la langue naturelle et le bruit introduit par les cas du type 5.4.b rendent inefficace l'utilisation de cette approche naïve pour la traduction.

Algorithme 5.2 : Synchronisation dans un arbre de comptage pour insertion

Entrées : Un vecteur de comptage (counts), l'arbre de comptage T, et l'alphabet A
Sorties : L'indice i du symbole courant dans l'alphabet, le nœud courant et le nœud parent où l'insertion doit commencer

```
1  i ← 0
2  current ← T
3  parent ← nil
4  tant que current.index ≠ nil ∨ i > current.index faire
5      si i < current.index alors
6          si counts[A[i]] = 0 alors
7              i ← i + 1
8          sinon
9              interrompre
10         fin
11     sinon
12         n ← current.children[counts[A[i]]]
13         si n ≠ nil alors
14             parent ← current
15             current ← n
16             i ← i + 1
17         sinon
18             interrompre
19         fin
20     fin
21 fin
22 retourner ⟨current, parent, i⟩
```

	This proportion seems correct.	:	*This assertion seems correct.*
(a)	↕	: :	↕
	Cette proportion semble correcte.	:	*Cette assertion semble correcte.*
	This one is incorrect.	:	*This one is not correct.*
(b)	↕	: :	↕
	Celle-ci est incorrecte.	:	*Celle-ci est **not correcte**.*

EXEMPLE 5.4 – Proportions analogiques croisées

Algorithme 5.3 : Insertion du sous-arbre correspondant
Une légère correction a été ajoutée par rapport à la version de [Langlais et Yvon, 2008], voir les lignes 12 à 15.

Entrées : Un vecteur de comptage (counts), le nœud courant, l'indice ic du symbole associé dans l'alphabet, l'alphabet A et le nœud parent
Sorties : Le nouveau nœud courant et son parent après insertion d'un nœud

```
1  count ← counts[A[ic]]
2  si current.index = nil alors
3      si count = 0 alors
4          current.index ← ic
5          n ← ⟨index = nil, forms = [], children = []⟩
6          current.children[count] ← n
7          parent ← current
8          current ← n
9      fin
10 sinon si current.index = ic alors
11     n ← ⟨index = nil, forms = [], children = []⟩
12     si count = 0 alors
13         n.forms ← current.forms
14         current.forms ← []
15     fin
16     current.children[count] ← n
17     parent ← current
18     current ← n
19 sinon si count ≠ 0 alors
20     n1 ← ⟨index = ic, forms = [], children = []⟩
21     n1.children[0] ← current
22     n2 ← ⟨index = nil, forms = [], children = []⟩
23     n1.children[count] ← n2
24     si parent ≠ nil alors
25         pour i : parent.children[i] = current faire
26             parent.children[i] ← n1
27         fin
28     fin
29 fin
30 retourner < current, parent >
```

Algorithme 5.4 : Recherche de quadruplets avec arbre de comptage

Entrées : Une paire de séquences $P_{in} = (s^1_{in}, s^2_{in})$ et un arbre de comptage T sur l'alphabet A
Sorties : L'ensemble des paires de séquences $P_{out} = (s^1_{out}, s^2_{out})$ indexées dans T telles que $comptage(P_{in}) = comptage(P_{out})$

$counts \leftarrow encode(s^1_{in}, s^2_{in})$
si $\exists\, symbol \in counts : symbol \notin lexicon(T)$ **alors**
  retourner $\{\}$
fin
$frontier \leftarrow \{(root(T), root(T))\}\,; i \leftarrow 1$
tant que $i \leq |A| \wedge frontier \neq \{\}$ **faire**
  $res \leftarrow \{\}\,; count \leftarrow counts[A[i]]$
  pour tous les $(n_1, n_2) \in frontier$ **faire**
    si $n_1.index = n_2.index = i$ **alors**
      pour chaque $child_1 \in n_1.children$ **faire**
        pour chaque $child_2 \in n_2.children$ **faire**
          si $child_1.count + child_2.count = count$ **alors**
            $res \leftarrow res \cup \{(child_1, child_2)\}$
          fin
        fin
      fin
      $child_1 \leftarrow n_1.children[count]\,; child_2 \leftarrow n_2.children[count]$
      si $child_1 \neq$ **nil** $\wedge\, |n_2.forms| > 0$ **alors**
        $res \leftarrow res \cup \{(child_1, n_2)\}$
      fin
      si $child_2 \neq$ **nil** $\wedge\, |n_1.forms| > 0$ **alors**
        $res \leftarrow res \cup \{(n_1, child_2)\}$
      fin
    sinon si $n_1.index = i$ **alors**
      $s \leftarrow n_1.children[count]$
      si $s \neq$ **nil alors**
        $res \leftarrow res \cup \{(s, n_2)\}$
      fin
      si $count = 0 \wedge |n_1.forms| > 0$ **alors**
        $res \leftarrow res \cup \{(n_1, n_2)\}$
      fin
    sinon si $n_2.index = i$ **alors**
      $s \leftarrow n_2.children[count]$
      si $s \neq$ **nil alors**
        $res \leftarrow res \cup \{(n_1, s)\}$
      fin
      si $count = 0 \wedge |n_2.forms| > 0$ **alors**
        $res \leftarrow res \cup \{(n_1, n_2)\}$
      fin
    sinon si $count = 0$ **alors**
      $res \leftarrow res \cup \{(n_1, n_2)\}$
    fin
  fin
  $frontier \leftarrow res$
  $i \leftarrow i + 1$
fin
retourner $\{(f_1, f_2) : f_1 \in p_1.forms, f_2 \in p2.forms, (p_1, p_2) \in frontier\}$

(a)

Affiche `fichier.txt` jusqu'à la ligne 7	:	Affiche `tab.csv` jusqu'à la ligne 50
↕	::	↕
`head fichier.txt -n 7`	:	`head tab.csv -n 50`

(b)

Compte les lignes du fichier `foo.c`	:	Compte les lignes du fichier C `bar`	::	Compile le fichier `foo.c`	:	Compile le fichier C `bar`
↕		↕		↕		↕
`wc -l foo.c`	:	`wc -l bar.c`	::	`gcc foo.c`	:	`gcc bar.c`

EXEMPLE 5.5 – Proportions analogiques directe (a) et indirecte (b) entre le français et le langage bash

En revanche, les homographes sont beaucoup plus courants dans le cadre du transfert entre langue naturelle et langage formel[6] . Les énoncés en langue naturelle qui correspondent à des requêtes contiennent des indications précises, avec notamment les paramètres de commandes tels que des valeurs numériques, noms de fichiers ou d'autres chaînes de caractères spécifiques.

L'exemple 5.5.a montre une proportion analogique croisée entre le français et le langage `bash`. On peut raisonnablement formuler l'hypothèse que les répétitions d'une même expression telle que "Affiche X jusqu'à la ligne N" sont fréquentes lors de l'interaction dialogique avec un assistant opérationnel, qu'il soit logiciel ou humain. Cependant, l'utilisation seule d'analogies directes pour répondre à l'utilisateur ne produirait de solution que dans ces cas de figure, c'est-à-dire lorsque le modèle de l'expression existe déjà dans la base d'exemples. Ce n'est pas suffisant compte tenu de la variabilité de la langue naturelle, qui permet de produire de multiples énoncés pour un même contenu sémantique. Le nombre de reformulations théoriquement possibles croît exponentiellement avec la taille de l'énoncé. Or, la méthode analogique indirecte décrite plus haut est plus souple quant aux reformulations. Elle permet en effet de trouver des solutions pour un énoncé dont le modèle n'est pas présent dans la base d'exemples. En contrepartie, elle requiert que tous les tokens de l'énoncé à transférer soient présents dans la base.

Le tableau ci-desous résume les propriétés de chacune des méthodes de résolution.

	direct	indirect
nouveaux tokens	✓	✗
nouveaux modèles	✗	✓

Il indique lorsque les méthodes peuvent retourner une solution, et lorsqu'elles ne le peuvent pas. Les deux méthodes peuvent donc être combinées afin d'en exploiter la complémentarité. La sous-section suivante présente les expériences menées avec un système les utilisant de manière parallèle.

5.5 Génération par analogie

Les proportions analogiques entre séquences requièrent des contraintes fortes quant aux sous-séquences contenues dans chacun des éléments du quadruplet. L'approche par transfert direct nécessite l'existence dans la base du modèle de l'énoncé à transférer, et l'approche par transfert indirect impose que chacun des tokens ait déjà été rencontré une fois. Il arrive donc qu'il manque un énoncé pour constituer une proportion et effectuer le transfert, alors même que cet énoncé peut lui-même être résolu. Ces énoncés intermédiaires peuvent être obtenus par le calcul de la fermeture transitive analogique du

6. Cette remarque vaut également pour le transfert entre deux langages formels.

	B_s	B_c
1	Compte les lignes du fichier foo.c	`wc -l foo.c`
2	Compte les lignes du fichier C bar	`wc -l bar.c`
3	Compile foo.c	`gcc foo.c`
4	Décompresse le fichier f.tar.gz	`tar xvz < f.tar.gz`
5	Décompresse f.tar.gz	`tar xvz < f.tar.gz`
6	Compile le fichier foo.c	`gcc foo.c`

EXEMPLE 5.6 – Base d'associations B et extension par fermeture transitive analogique (ajout de l'association 6)

langage que représente la base d'exemples. La **fermeture transitive analogique*** α d'un langage Λ est définie comme suit [Lepage et Denoual, 2005] :

$$\alpha(\Lambda,\Lambda,\Lambda) = \left\{t | \exists\,(x,y,z) \in \Lambda^3, [x : y :: z : t]\right\}$$

α est une opération ternaire car elle associe à Λ l'ensemble des phrases obtenues en résolvant toutes les équations analogiques formées à partir des phrases de Λ. On appellera fermeture transitive d'ordre n d'un langage Λ l'ensemble défini par :

$$\begin{aligned}\Lambda_n &= \alpha(\Lambda_{n-1},\Lambda_{n-1},\Lambda_{n-1}) \cup \Lambda_{n-1} \\ &= \alpha(\Lambda_{n-1},\Lambda_{n-1},\Lambda_{n-1})\end{aligned}$$

L'algorithme récursif pour le transfert indirect que nous avons décrit dans la section 5.3.1 explore implicitement le domaine des fermetures transitives analogiques d'ordre arbitraire. En effet, la traduction d'un énoncé peut s'appuyer non seulement sur un triplet d'énoncés source de la base mais aussi sur un triplet comprenant un énoncé source lui-même généré par analogie au cours d'une itération antérieure. Cet énoncé généré par analogie n'appartient pas nécessairement à la base, il s'agit d'un élément de Λ_n où n est le niveau de la récursion.

Pour exemple, considérons la base minimale présentée dans la figure 5.6 (associations 1 à 5) pour transférer la requête e_u^s = "Compile le fichier C bar", reprise de l'exemple précédent. La base d'exemples proposée ne contient pas d'associations multiples pour limiter la combinatoire de l'exemple. La base ne permet pas de constituer un triplet (e_i^s, e_j^s, e_k^s) tel que $[e_i^s : e_j^s :: e_k^s : e_u^s]$ soit une proportion valide. Il est cependant possible de résoudre $[e_5^s : e_3^s :: e_4^s : ?]$ dans B_s ainsi que l'équation associée dans B_c $[e_5^c : e_3^c :: e_4^c : ?]$ pour obtenir une nouvelle association appartenant à la fermeture transitive d'ordre 1 :

Cet ajout à la base d'exemples permet ensuite de former la proportion source $[e_1^s : e_2^s :: e_6^s : e_u^s]$ et de résoudre l'équation cible correspondante $[e_1^t : e_2^t :: e_6^t : ?]$ pour obtenir la solution e_u^t "`gcc bar.c`".

L'algorithme de résolution récursif tel que nous venons de le dérouler restreint le nombre d'énoncés générés utilisés et leur position. Un seul énoncé provenant de Λ_{n+1} peut être placé dans la proportion source, et seulement en troisième position (e_k^s). Cela interdit *a priori* les transferts requérant un énoncé généré en première position (e_i^s) ou bien nécessitant plus d'un énoncé généré. Notons cependant qu'il s'agit d'un algorithme "paresseux", qui est supposé faire tendre asymptotiquement la base de connaissances vers l'ensemble Λ^* des formes réellement utilisées. Lepage [Lepage et Denoual, 2005] précise en effet que chaque énoncé généré pour les besoins de la récursion est ensuite ajouté à la base d'exemples. Cependant, cet incrément dépend fortement des proportions qui sont constituées au cours de l'utilisation du système.

	R	bash1 (partielle)	bash1	bash2
ensemble initial				
... total associations	448	435	435	501
... associations uniques	448	420	420	469
... total requêtes uniques	424	420	420	468
... total commandes uniques (en-têtes)	203	20 (13)	20 (13)	343 (93)
ensemble généré				
... total associations	452 419	39 089	736 864	687 247
... associations nouvelles	52 634	36 719	355 114	187 368
... associations uniques	18 031	5 379	40 853	25 780
... associations nouvelles et uniques	17 585	5 227	40 484	25 397
... total requêtes uniques	13 947	5 335	40 738	25 377
... total commandes uniques (en-têtes)	4 069	21 (13)	55 (13)	4 713 (97)

TABLEAU 5.1 – Génération analogique systématique appliquée aux corpus collectés
Note 1 : les en-têtes des commandes ne sont pas comptabilisées pour le langage R car sa structure rend cette propriété non pertinente, contrairement à bash.
Note 2 : la colonne bash1 partielle décrit les résultats de la génération utilisés dans l'article [Letard *et al.*, 2015]. En effet, l'algorithme de génération employé à ce moment était sous-optimal et a dû être interrompu faute de temps.

Une alternative à cette méthode de génération à la demande est l'**extension de la base par génération analogique**[*], elle consiste à générer en extension l'ensemble Λ_1 à partir de Λ_0, pour ensuite y appliquer les méthodes de transfert classiques et un peu plus légères (*i.e.* les méthodes directe et indirecte telle que nous les avons décrites). Il s'agit d'un processus coûteux, nécessitant $\mathrm{O}\left(|B|^3\right)$ opérations, et n'est donc applicable qu'à partir d'une base d'exemples restreinte. La génération en extension s'exécute entièrement hors ligne, mais occasionne par la suite un surcoût computationnel pour l'algorithme de transfert dont la complexité dépend de la taille de la base. En revanche, à l'inverse de la méthode récursive seule, elle permet de couvrir les cas dans lesquels la résolution de l'énoncé e_u^s dépend d'une proportion analogique requérant plus d'un énoncé provenant de Λ_1, ou en première ou deuxième position.

Nous avons appliqué l'extension par génération analogique aux bases d'exemples présentées en section 2.4. Le tableau 5.1 décrit la distribution du contenu des sous-ensembles utilisés pour la génération, ainsi que le contenu des ensembles obtenus. Seuls les ensembles d'exemples dits d'entraînement ont été utilisés pour générer de nouveaux exemples.

Les requêtes générées sont nombreuses et entrent dans les catégories suivantes :

1. déjà présente dans l'ensemble initial
2. nouvelle et sémantiquement cohérente [7]
3. nouvelle et syntaxiquement incorrecte mais compréhensible
4. nouvelle et syntaxiquement correcte mais sémantiquement incohérente
5. nouvelle et sémantiquement incohérente

7. On définit une requête comme sémantiquement cohérente si le kappa de Cohen [Cohen, 1960] obtenu pour son association avec une commande par de multiples participants est supérieur à une constante : $\kappa > c \geq 0$. La constante c est à déterminer par comparaison avec les valeurs de kappa obtenues pour des requêtes rédigées par des humains.

catégorie	exemple de génération
2	peux-tu afficher le contenu du fichier foo s'il-te-plaît
3	affiche me le fichier foo
4	écrit foo s'il te plaît
5	affecte 42 égale à la variable FOO si elle n'est pas

EXEMPLE 5.7 – Échantillon de requêtes générées par transitivité analogique

L'exemple 5.7 donne un échantillon de requêtes générées pour les quatre dernières catégories. Rappelons que l'objectif ici est avant tout la production de nouvelles associations pour la base d'exemples. Les requêtes produites n'ont pas vocation à être lues, elles seront considérées comme satisfaisantes si elles permettent potentiellement de résoudre une requête de l'utilisateur en participant à la constitution d'un triplet du domaine source.

Le processus d'extension de la base nécessite également de produire une commande correspondant à chacune de ces requêtes générées. Pour ce faire, on transfère dans le domaine cible les équations analogiques du domaine source de manière similaire à l'algorithme de transfert. Pour une équation $[e_i^s : e_j^s :: e_k^s : ?]$ produisant la requête e_l^s, on résoud toutes les équations $[e_{i,l}^t : e_{j,m}^t :: e_{k,n}^t : ?]$ telles que $\left(e_i^t, e_{i,l}^t\right), \left(e_j^s, e_{j,m}^t\right), \left(e_k^s, e_{k,n}^t\right) \in B$. Les commandes ainsi obtenues sont associées à e_l^s dans la base étendue produite.

En pratique, le problème de la cohérence des commandes générées ne s'est pas posé, puisque toutes les équations $[e_{i,l}^t : e_{j,m}^t :: e_{k,n}^t : ?]$ qui ont produit des solutions étaient des équations canoniques du type $[a : a :: b : ?]$ ou $[a : b :: a : ?]$, dont la solution est triviale et ne nécessite aucune modification de la chaîne de caractères. Cette considération devient importante en revanche dès que le domaine cible est un langage naturel comprenant des variations nombreuses et des ambiguités.

Le facteur d'augmentation de la base d'exemples par fermeture transitive analogique est très élevé, donnant dans notre cas $\frac{\Lambda_1}{\Lambda_0} \simeq 12,26$. De plus, la taille du produit cartésien cubique de la base à énumérer pour constituer les équations analogiques à résoudre devient rapidement hors de portée ($5\ 000^3 > 10^{11}$). La génération de Λ_n pour des valeurs de n supérieures à 1 ou 2, selon la taille de la base initiale, n'est donc pas réaliste.

5.6 Protocole expérimental

Afin de mesurer la performance du transfert par analogie, nous avons mis en place un protocole permettant d'isoler chacun des paramètres que nous avons proposés : la combinaison du transfert direct et indirect ainsi que la génération en extension de la fermeture transitive analogique de la base d'exemples. Les hypothèses posées et que nous soumettons aux expériences concernent le cas du transfert depuis une langue naturelle vers un langage formel, les idées en ont été avancées dans les sections précédentes. Nous avons d'abord conjecturé une bonne complémentarité des approches par transfert direct et indirect. On s'attend donc à ce que leur utilisation conjointe permette une amélioration substancielle [8] des résultats par rapport au transfert indirect seul. La seconde conjecture est que l'extension de la base par fermeture transitive permet de compléter les bonnes réponses produites à l'aide de la seule base initiale.

8. Par amélioration substancielle, nous entendons une amélioration plus marquée pour le transfert de la langue naturelle vers un langage formel que pour une traduction entre deux langues naturelles, puisque l'hypothèse de complémentarité tient pour la première configuration et ne tient pas pour la seconde.

Les expériences ont d'abord été menées sur la base base `bash1`, puis nous les avons appliquées aux autres bases collectées. Toutes ces bases ont été introduites au chapitre 2.

Les tests ont été effectués pour `bash1` à l'aide de deux ensembles différents, chacun comportant 77 requêtes. Le premier est extrait de la base d'exemples initiale [9], il contient donc les mêmes paramètres de requêtes que celle-ci ainsi que des formulations plutôt similaires des requêtes. Nous l'appelons l'ensemble QUOTIDIEN, car il s'apparente au cas dans lequel les utilisateurs interagissant avec le système ont déjà interagi auparavant, et contribué à l'élaboration de ses connaissances. Le deuxième ensemble de test est constitué de requêtes qui n'ont pas été collectées auprès des rédacteurs de la base principale. Elles ne sont donc pas influencées par la formulation des requêtes de la base de connaissances. En outre, les paramètres des requêtes ont été systématiquement modifiés. Nous appelons cet ensemble de test l'ensemble NOUVEAU, car il correspond par contraste au premier, au cas dans lequel un nouvel utilisateur est amené à interagir avec le système. L'utilisateur et le système n'ont donc pas pu s'influencer l'un l'autre. En d'autres termes, le contenu des connaissances du système est indépendant des exemples formulés par l'utilisateur, et la formulation des requêtes par l'utilisateur n'est pas influencée par son utilisation du système. Les tableaux de l'exemple 5.8 proposent un aperçu du contraste entre les exemples de chacun des ensembles de test.

QUOTIDIEN		
c'est quoi le fichier foo	→	`file foo`
compte les mots du fichier foo	→	`wc -w foo`
affiche-moi le contenu de foo	→	`cat foo`

NOUVEAU		
efface bla du dossier courant	→	`rm ./bla`
quel est le type du fichier truc	→	`file true`
montre moi le contenu du fichier blo	→	`cat blo`

EXEMPLE 5.8 – Échantillon des exemples contenus dans l'ensemble QUOTIDIEN et dans l'ensemble NOUVEAU associés à la base `bash1`

Nous avons également appliqué le même protocole sur les bases `bash2` et `R` afin de comparer les résultats respectifs à la lumière des différences constitutives des bases. Pour cela, nous avons testé les réponses à l'aide d'ensembles de test de 77 requêtes également, extraits des bases d'exemples concernées. La différence pour ces bases d'exemples est qu'elles n'ont été testées que sur un seul ensemble de test. La raison pour `R` est que la source de la collecte est unique, et le nombre d'exemples pour chaque commande est quasiment réduit à 1. En ce qui concerne la base `bash2`, la distinction pour les tests entre un ensemble de requêtes dit "quotidien" et un ensemble dit "nouveau" n'est pas pertinent car le protocole de collecte inclut les formulations toutes indépendantes de plusieurs utilisateurs. Un sous-ensemble de requêtes utilisé en tant que test sera donc toujours plutôt "nouveau" en général.

Lors des tests, nous mesurons principalement la proportion de réponses correctes, ou la précision du système. Trois options se posent quant à la prise en compte des vrais négatifs, c'est-à-dire des cas dans lesquels le système ne donne pas de réponse lorsqu'effectivement aucune réponse correcte ne

9. La base initiale est donc divisée en un ensemble de 435 exemples utilisé comme base d'exemples, et un ensemble de 77 exemples utilisé pour le test.

pouvait être trouvée. On peut les considérer comme des réponses correctes, incorrectes, ou bien les exclure du score global. La première possibilité rend le score représentatif de la capacité du système à proposer la réponse théorique correcte accessible par rapport à la base d'exemples disponible. Si l'on évalue le système pour sa capacité à répondre "aussi bien qu'un humain", cette première option n'est probablement pas pertinente, car l'humain peut raisonner selon d'autres principes que l'analogie formelle et qui lui permettraient d'obtenir une réponse correcte. Dans ce cas, on ne peut considérer comme systématiquement corrects les silences pour des réponses inaccessibles par analogies. Notons que la comparaison effective avec l'être humain n'est pas applicable dans notre cadre, car il suppose un très fort déséquilibre entre le système, limité à un ensemble de connaissances strictement limité, et l'humain qu'on ne peut empêcher/qui ne peut s'empêcher d'utiliser implicitement des connaissances extérieures à la base d'exemples à sa disposition. La seconde option, considérer les non-réponses comme incorrectes, est intéressante dans une perspective applicative, où on s'intéresse notamment à maximiser le nombre de réponses. La dernière option consistant à ignorer les silences du score global semble finalement la plus neutre, ne prenant pas parti. C'est donc celle que nous avons appliquée.

5.7 Résultats

5.7.1 Base d'exemples `bash1`

Les tableaux 5.2 et 5.3 montrent respectivement les résultats obtenus pour les tests sur l'ensemble QUOTIDIEN et sur l'ensemble NOUVEAU, sans application de la génération par analogie. La dernière colonne (match exact), reportant les performances en s'appuyant uniquement sur la correspondance exacte d'un exemple du corpus à la requête de l'utilisateur, a été ajoutée pour comparaison. Les requêtes qui sont traitées par correspondance exacte sont également trivialement couvertes par l'approche directe et par l'approche indirecte (à l'aide d'analogies canoniques par exemple). Le score obtenu en utilisant exclusivement les analogies indirectes indique à quel point l'ensemble de test et la base d'exemples sont similaires en termes de vocabulaire (paramètres des requêtes compris). En effet, aucune proportion respectant le comptage des tokens ne peut être constituée si l'un des tokens de la requête est absent de la base d'exemples. Les tests sur l'ensemble QUOTIDIEN soulignent clairement la proximité de cet ensemble à la base d'exemples avec les 47% de bonnes réponses données. L'absence de requêtes couvertes par la méthode directe pour l'ensemble NOUVEAU atteste quant à elle de la nouveauté du vocabulaire utilisé dans cet ensemble de test, et en particulier de la variation systématique des paramètres.

Le score obtenu grâce aux analogies directes seules rend compte de la proportion des requêtes de test qui sont identiques à au moins une requête du corpus à l'exception éventuelle de leurs paramètres. Les 4% obtenus sur QUOTIDIEN attestent de la variation linguistique introduite lors de la collecte de la base globale. Les scores reportés sont des bornes inférieures du nombre de proportions analogiques existantes pour résoudre les requêtes de test, car les mauvaises solutions à des équations analogiques sont également des sources d'erreurs possibles. Cependant, ces mauvaises solutions analogiques sont rares, comme en témoignent les très faibles différences entre le ratio de réponses données et le score considérant tous les silences comme des erreurs.

Par ailleurs, on peut constater que le score des méthodes directe et indirecte utilisées ensemble ne correspond pas exactement à la somme des scores de chacune des méthodes prises séparément. Cela s'explique par l'existence de certaines requêtes traitées par chacune des méthodes. Les cas les plus triviaux et les plus nombreux de cette situation sont les requêtes également résolues par correspondance exacte. Les autres cas, plus rares, peuvent être illustrés par l'exemple 5.9.

Le token "ps" est consiéré comme un paramètre lors de la résolution directe, mais comme un mot du lexique lors de la résolution indirecte. Cela contraint pour le cas indirect à trouver dans la base trois

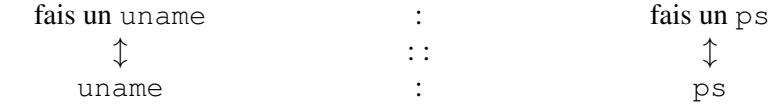
	indirect	direct	ensemble	match exact
ratio de réponses données	$0,47$	$0,05$	$0,47$	$0,04$
score (silence = faux)	$0,47$	$0,04$	$0,47$	$0,04$
score (silence ignoré)	1	$0,75$	1	1
temps par requête min	$0,008$	$0,007$	$0,008$	$0,005$
temps par requête max	$8,254$	$0,233$	$8,381$	$0,008$
temps par requête moyen	$0,705$	$0,097$	$0,813$	$0,005$
temps par requête médian	$0,372$	$0,071$	$0,415$	$0,005$
écart-type du temps par requête	$1,502$	$0,070$	$1,508$	$0,001$

TABLEAU 5.2 – Résultats des tests sur l'ensemble QUOTIDIEN pour la base `bash1`

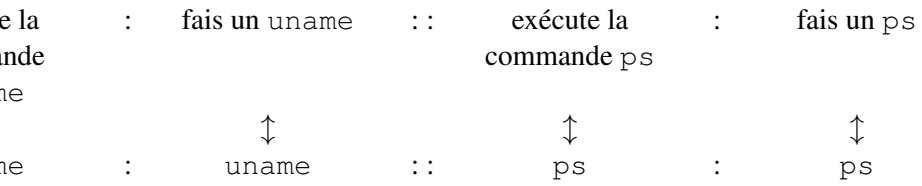
	indirect	direct	ensemble	match exact
ratio de réponses données	0	$0,12$	$0,12$	0
score (silence = faux)	0	$0,12$	$0,12$	0
score (silence ignoré)	–	1	1	–
temps par requête min	$0,029$	$0,019$	$0,047$	$0,005$
temps par requête max	$0,322$	$0,238$	$0,379$	$0,007$
temps par requête moyen	$0,047$	$0,101$	$0,145$	$0,005$
temps par requête médian	$0,039$	$0,073$	$0,108$	$0,005$
écart-type du temps par requête	$0,044$	$0,070$	$0,082$	$< 0,001$

TABLEAU 5.3 – Résultats des tests sur l'ensemble NOUVEAU pour la base `bash1`

(a)



(b)



EXEMPLE 5.9 – Un cas de transfert direct (a) et un autre indirect (b) donnant la même réponse correcte à la requête "fais un `ps`".

	indirect	direct	ensemble	match exact
ratio de réponses données	0,53	0,49	0,56	0,47
score (silence = faux)	0,52	0,47	0,53	0,47
score (silence ignoré)	0,98	0,95	0,95	1
temps par requête min	0,023	0,021	0,028	0,017
temps par requête max	2 060,227	28,053	2 369,983	0,036
temps par requête moyen	124,433	4,438	138,565	0,024
temps par requête médian	2,706	0,768	5,343	0,022
écart-type du temps par requête	355,602	7,584	376,550	0,005

TABLEAU 5.4 – Résultats sur l'ensemble QUOTIDIEN pour la base `bash1` avec génération de la fermeture transitive

	indirect	direct	ensemble	match exact
ratio de réponses données	0,01	0,23	0,25	0
score (silence = faux)	0,01	0,19	0,21	0
score (silence ignoré)	1	0,83	0,84	–
temps par requête min	2,248	0,501	2,687	0,015
temps par requête max	3 699,076	29,180	3 773,472	0,026
temps par requête moyen	54,999	7,860	63,043	0,019
temps par requête médian	3,169	4,578	7,714	0,019
écart-type du temps par requête	422,051	7,922	429,431	0,002

TABLEAU 5.5 – Résultats sur l'ensemble NOUVEAU pour la base `bash1` avec génération de la fermeture transitive

autres requêtes dont au moins l'une utilise déjà ce terme.

Le tableau 5.2 montre de mauvais résultats pour le transfert analogique direct. En effet, la base d'exemples globale collectée (donc incluant l'ensemble de test QUOTIDIEN) ne comprend aucune variation sur les seuls paramètres de la requêtes. La seule réponse correcte provient en réalité d'une correspondance exacte fortuite. Le transfert analogique indirect, quant à lui, donne de meilleurs résultats que sur l'ensemble NOUVEAU (aucune réponse n'est donnée sur l'ensemble NOUVEAU) pour deux raisons : d'une part, les variations linguistiques présentes dans l'ensemble de test et dans la base ont été introduites lors de la même collecte par les mêmes personnes, ce qui explique la complémentarité analogique que traduit le score ; d'autre part, aucun paramètre des requêtes de QUOTIDIEN n'a été modifié, ce qui n'est pas le cas pour NOUVEAU et explique l'absence de correspondances analogiques à cause du comptage des tokens.

Le tableau 5.3 montre une chute drastique dans les performance de la méthode indirecte due, nous l'avons vu, à la variation systématique des paramètres des requêtes, qui introduit des tokens nouveaux dans chacune d'elles. Par ailleurs, le nombre de requêtes traitées correctement par analogie directe est plus élevé que pour QUOTIDIEN. On peut en déduire que certaines requêtes ont été indépendamment saisies de manière identique, à l'exception de leurs paramètres, par des rédacteurs différents au cours des collectes de la base globale et de l'ensemble NOUVEAU. L'absence de correspondance exacte est due à la même raison que l'absence d'analogies indirectes.

L'ajout à la base d'exemples générés automatiquement a un effet notable sur les correspondances exactes, comme le montre le tableau 5.4. Le score par match exact est en réalité celui de la méthode indirecte donné dans le tableau 5.2. En effet, lorsqu'une analogie proportionnelle peut être formée à partir d'un triplet de requêtes de la base ainsi que la requête de l'utilisateur, alors ce même triplet peut servir à générer la requête de l'utilisateur avant qu'il ne la soumette. Les scores en utilisant exclusivement la méthode directe ou la méthode indirecte sont sensiblement plus élevés que dans le tableau 5.2 sans génération, mais cette augmentation est en grande partie due aux correspondances directes, qui sont également identifiées par la méthode directe et la méthode indirecte. Autrement dit, la partie intéressante du score de 52% obtenue par méthode indirecte tient en réalité à $52 - 47 = 5\%$ de *nouvelles* réponses correctes réellement trouvées grâce au raisonnement analogique indirect sur la base générée. Le raisonnement direct quant à lui, ne produit aucune bonne réponse de plus que ce qui a déjà été pré-calculé par l'opération de génération. Ce dernier point était attendu car tous les paramètres des requêtes et commandes de l'ensemble QUOTIDIEN sont déjà présentes dans la base d'exemples.

Enfin, le tableau 5.5 montre les mêmes faibles résultats pour les correspondances exactes et la méthode indirecte qu'observés dans le tableau 5.3. On constate une progression concernant le score de la méthode directe. On peut l'expliquer par le fait que la génération aura produit des requêtes respectant le même motif que certaines requêtes de l'ensemble de test, avec pour seule différence les valeurs de leurs paramètres. En effet, les paramètres sont les seuls éléments qui ne peuvent être inclus dans les énoncés générés s'ils ne sont pas déjà présents dans la base d'exemples.

Concernant la durée d'exécution, elle est en général très variable d'une requête à l'autre. En effet, une correspondance exacte sera identifiée presque instantanément, une correspondance directe en temps linéaire et une correspondance indirecte en temps variable selon la constitution de la base et la requête soumise. Les requêtes très similaires à de nombreux exemples de la base en terme de comptage des tokens causent les calculs les plus longs. Pour les cas dans lesquels aucune solution n'est trouvée, s'il s'agit d'un token de la requête absent du corpus, la durée n'est que linéaire afin de vérifier les analogies directes ; si en revanche tous les tokens de la requête appartiennent au corpus, il faut compter la même durée que pour une tentative de transfert indirect.

Notons que les résultats diffèrent de ceux présentés dans [Letard *et al.*, 2015] à cause de la phase de fermeture transitive analogique de la base d'exemples qui n'avait pas été exhaustive pour l'article (voir note du tableau 5.1). Il est intéressant de constater qu'outre le temps d'exécution qui a passablement souffert de l'augmentation de la taille de la base, les scores n'ont que peu varié. Seules quelques réponses ont été ajoutées, certaines erronnées. La conjecture que l'on peut en tirer est que, bien que toutes les associations requête-commande de la base générée soient uniques, la base finale contient une grande part de redondance pour le raisonnement analogique. Il reste à démontrer si cette redondance provient essentiellement de la méthode analogique de génération ou bien de la redondance analogique préalable de la base `bash1` en particulier (*cf.* section 2.4.2).

5.7.2 Comparaison avec les bases `R` et `bash2`

Les résultats obtenus avec les bases `R` et `bash2` sont respectivement consignés dans les tableaux 5.6 et 5.7. La couverture sur ces bases en termes de nombre de réponses données par le système est moins large que sur la base d'exemples `bash1` testée sur l'ensemble QUOTIDIEN. Elle avoisine 30% pour chacune des deux bases, contre 47% sur `bash1`. Rappelons que les bases `R` et `bash2` ne disposent que d'un seul ensemble de test chacune, directement extrait d'un sous-ensemble d'exemples, contrairement à `bash1` qui a également été testée avec un ensemble moins biaisé comme nous l'avons vu. Cependant, les conditions de collecte de `bash2` auprès d'un plus grand nombre de participants et en minimisant les contraintes et les influences (*cf.* chapitre 2) lui confèrent une meilleure représentativité de l'utilisation probable du système ainsi qu'une grande variabilité. En conséquence, l'ensemble de test extrait des exemples de `bash2` peut être considéré comme plus proche de NOUVEAU que de QUO-

	indirect	direct	ensemble	match exact
ratio de réponses données	$0,27$	$0,12$	$0,29$	$0,08$
score (silence = faux)	$0,26$	$0,10$	$0,27$	$0,06$
score (silence ignoré)	$0,95$	$0,88$	$0,95$	$0,83$
temps par requête min	$0,005$	$0,006$	$0,006$	$0,003$
temps par requête max	$1,355$	$0,290$	$0,290$	$0,010$
temps par requête moyen	$0,475$	$0,159$	$0,159$	$0,006$
temps par requête médian	$0,554$	$0,172$	$0,172$	$0,005$
écart-type du temps par requête	$0,402$	$0,075$	$0,075$	$0,001$

TABLEAU 5.6 – Résultats avec la base d'exemples `R`

	indirect	direct	ensemble	match exact
ratio de réponses données	$0,09$	$0,25$	$0,31$	$0,01$
score (silence = faux)	$0,09$	$0,17$	$0,25$	$0,01$
score (silence ignoré)	1	$0,68$	$0,79$	1
temps par requête min	$0,007$	$0,006$	$0,006$	$0,003$
temps par requête max	$1,792$	$0,123$	$2,004$	$0,005$
temps par requête moyen	$0,318$	$0,056$	$0,409$	$0,004$
temps par requête médian	$0,046$	$0,054$	$0,112$	$0,004$
écart-type du temps par requête	$0,424$	$0,023$	$0,473$	$< 0,001$

TABLEAU 5.7 – Résultats avec la base d'exemples `bash2`

TIDIEN, pour reprendre les dénominations de `bash1`, comme en atteste la distribution du nombre de réponses entre la méthode directe et méthode indirecte. Cette proximité peut expliquer la couverture plus faible sur `bash2`, comparée à celle sur `bash1` testé avec QUOTIDIEN. Elle reste néanmoins supérieure à la couverture sur `bash1` testé avec NOUVEAU. Cela peut être dû à la différence de constitution des ensembles de test. En effet, l'ensemble de test de `bash2` contient des requêtes formulées par les mêmes participants que celles de la base d'exemples. Cependant, il nous semble hâtif de s'en tenir à cette seule explication, car on peut s'attendre à ce qu'un ensemble de test collecté auprès d'une autre personne de la même communauté engendre des résultats similaires, grâce à la variabilité contenue dans la base d'exemples elle-même.

La distribution des réponses données pour le test sur la base d'exemples `R` est plus équilibrée que pour les autres tests, excepté bien entendu ceux sur la fermeture transitive. On peut l'expliquer par l'orientation de la collecte de manière à équilibrer les formulations afin de produire un échantillon d'apprentissage varié (*cf.* section 2.4.1). En effet, la répétition dans la base d'une même requête pour plusieurs commandes différentes mais sémantiquement équivalentes donne lieu à des correspondances exactes lors de la résolution. De plus, les variations dans la seule formulation des requêtes donnent lieu à des réponses trouvées par transfert analogique indirect. Enfin, les variations sur les paramètres seuls permettent de trouver des réponses par transfert analogique direct. En revanche, les différentes méthodes donnent des réponses pour les mêmes requêtes, comme on peut le lire en deuxième ligne du tableau 5.6 : le score des méthodes ensemble est plus proche du maximum des deux scores séparés que de leur somme. Les tests sur la base `bash2` quant à eux, montrent une faible superposition de la couverture des requêtes par les méthodes de résolution (*cf.* deuxième ligne du tableau 5.7), et donc une complémentarité accrue entre le transfert analogique direct et le transfert analogique indirect. On peut

	indirect	direct	ensemble	match exact
ratio de réponses données	0,32	0,32	0,35	0,27
score (silence = faux)	0,25	0,21	0,29	0,19
score (silence ignoré)	0,76	0,64	0,81	0,71
temps par requête min	0,011	0,016	0,007	0,009
temps par requête max	256,028	18,686	261,745	0,027
temps par requête moyen	49,621	6,131	49,269	0,013
temps par requête médian	1,313	2,961	16,350	0,012
écart-type du temps par requête	62,790	6,716	57,412	0,004

TABLEAU 5.8 – Résultats avec la base d'exemples `R` augmentée par génération de la fermeture transitive

	indirect	direct	ensemble	match exact
ratio de réponses données	0,09	0,34	0,35	0,07
score (silence = faux)	0,09	0,17	0,18	0,07
score (silence ignoré)	1	0,50	0,52	1
temps par requête min	0,022	0,070	0,019	0,009
temps par requête max	10 884,791	4,893	10 829,109	0,027
temps par requête moyen	217,418	2,962	220,388	0,013
temps par requête médian	1,960	3,644	6,091	0,012
écart-type du temps par requête	1 237,992	1,550	1 230,670	0,004

TABLEAU 5.9 – Résultats avec la base d'exemples `bash2` augmentée par génération de la fermeture transitive

l'attribuer à la variation naturellement incluse dans la base d'exemples collectée (*cf.* section 2.4.3).

Concernant la précision du système sur ces tests (*cf.* troisième ligne des tableaux), elle est dans l'ensemble moins bonne que sur la base `bash1`. L'effet est assez peu marqué pour la base `R`, mais plus important pour `bash2`. On remarque cependant que l'essentiel des mauvaises réponses données sont dues à une seule analogie. Il s'agit d'une analogie directe fondée sur l'association dans la base de la requête "ls" à la commande "`ls`". Le raisonnement analogique formel autorise ainsi toute requête composée d'un unique terme à être associée à la commande composée exactement de ce même terme, ce qui conduit évidemment à des erreurs. Elles peuvent être évitées par l'interdiction *ad hoc* de toute analogie directe opérant un transfert complet des termes de la requête vers la commande.

Les tableaux 5.8 et 5.9 contiennent les résultats après extension des bases d'exemples respectives par fermeture transitive analogique. On y constate une augmentation mesurée de la proportion de réponses données. Le nombre absolu de bonnes réponses croît dans le cas de la base `R`, mais décroît pour `bash2` lorsque les méthodes sont mises en commun. De même que sur la base `bash1`, le délai de réponse par requête augmente au delà du raisonnable pour une interaction "instantanée" homme-machine.

5.7.3 Remarque sur les indicateurs considérés

L'évaluation d'un système de traduction se fait souvent au moyen de la F-mesure, plus spécifiquement F1, qui agrège précision et rappel pour donner une estimation unifiée de la performance du système. La précision est la proportion de bonnes réponses présentes dans l'ensemble des réponses

données, le rappel est la proportion de réponses données dans l'ensemble des réponses correctes possibles, qu'on appelle aussi l'ensemble des références. La mesure du rappel n'est pas applicable, ou non pertinente, dans le cas du transfert depuis la langue naturelle vers un langage formel. En effet, l'ensemble des références pour chaque exemple est réduit à 1 sauf exception, et dans tous les cas, le système ne produit qu'une seule réponse. Par transitivité, la F-mesure également devient non pertinente.

Pour cette raison, nous examinons séparément les taux de réponse et scores du système, en distinguant entre un score tenant compte du silence comme d'une mauvaise réponse, et un score ignorant tous les cas de non réponse.

5.8 Conclusion

Les résultats obtenus confirment globalement les rôles respectifs des méthodes directe et indirecte de transfert par analogie. Les analogies directes permettent de traiter une nouvelle valeur quelconque pour tout paramètre des requêtes. Les tests sur l'ensemble NOUVEAU, avec variation systématique des paramètres, ont montré l'aspect indispensable de ce point pour le transfert depuis la langue naturelle vers le langage formel. Précisons qu'il ne s'agit pas d'un biais lié à la petite taille de la base de connaissances utilisée qui ne contiendrait pas encore suffisemment de variation : en effet, la valeur des paramètres ne peut pas être prédite car il peut arriver qu'elle résulte de tirages pseudo-aléatoires ; elle ne peut pas non plus être mémorisée étant donné que pour la raison précédente, le nombre de paramètres possibles d'une longueur n donnée est très grand. À titre d'exemple, on peut former 84^{50} URLs différentes de 50 caractères [10], et ce sans même inclure celles de longueurs inférieures. Une prédiction partielle de la forme des paramètres nécessiterait l'acquisition et l'exploitation d'un modèle plus élaboré de l'utilisateur et/ou du contexte d'exécution. Les analogies indirectes quant à elles, permettent la réorganisation et le mélange d'éléments linguistiques des requêtes. En contrepartie, elles rendent impossible le traitement d'énoncés contenant un mot absent du vocabulaire de la base d'exemples, y compris pour les paramètres des requêtes.

Ces deux stratégies sont donc complémentaires pour le transfert depuis la langue naturelle vers un langage formel. Les résultats obtenus pour `bash1` sur l'ensemble NOUVEAU suggèrent que pour le cas réel dans lequel les requêtes peuvent contenir des variations de paramètres, la performance du système dépend essentiellement du transfert direct, que ce soit avec la base d'exemples initiale ou une base enrichie par génération. Les tests sur la base `bash2` la moins contrainte confirment cette tendance. On peut supposer que le nombre de paramètres possibles est beaucoup plus grand que le nombre de formulations naturelles de requêtes, il est donc important de collecter ou générer autant de ces reformulations que possible, pourvu qu'elles correspondent dans l'ensemble à des usages réels.

Cependant, une grande partie des énoncés générés ne correspondent pas à des requêtes correctement formées en français. Nous ne l'avons pas estimé précisément à cause du coût d'annotation par rapport au très grand nombre d'énoncés générés. On peut conjecturer que cette grande proportion d'énoncés incohérents explique le gain relativement faible de bonnes réponses par rapport à la taille de la base générée. On calcule ainsi que le ratio entre les bonnes réponses apportées et le nombre d'exemples ajoutés à la base par génération évolue entre $0,0010$ et $0,0025$.

Afin de conserver une indépendance vis-à-vis des langages utilisés, nous n'avons pas mis de connaissances explicites sur la syntaxe de la langue naturelle ni du langage formel à disposition du système. Cela a permis notamment d'appliquer notre système sans aucune adaptation sur la base d'exemples `R`. Cependant, tous les langages formels sont basés sur une grammaire non ambigüe qui pourrait être

10. Il existe 84 caractères légaux dans l'écriture d'une URL.

utilisée pour améliorer le traitement analogique du côté du langage cible par une vérification de la validité des expressions, par exemple. Dans notre cas, ce n'était en réalité pas nécessaire, car la très grande majorité des analogies dans le domaine cible étaient des proportions canoniques, c'est-à-dire du type $[a : a :: b : b]$ ou $[a : b :: a : b]$. Cette propriété vient, d'une part, du fait que le nombre de commandes distinctes présentes dans le corpus était relativement petit, et d'autre part, du fait qu'en réalité, très peu de requêtes sont associées à plus d'une commande, ce qui aurait pour effet d'ajouter de l'ambiguïté.

Quelques uns des résultats que nous avons obtenus ne sont pas tout à fait probants et réclameraient une réplication avec des bases d'exemples et de test plus nombreuses et plus conséquentes, au sens du nombre d'associations réellement fournies par des humains, car les associations générées ne sont que très peu représentatives de l'usage. Les améliorations obtenues avec la base étendue par génération sont plutôt limitées. La pertinence de cette approche devant la mise en place d'une collecte à plus large échelle est donc en question, notamment au regard des forts coûts additionnels en temps d'exécution. Cependant, considérant l'exhaustivité de la recherche et la simplicité des équations analogiques du domaine cible, l'ajout de nouveaux exemples ne peut intuitivement pas conduire à une dégradation des réponses produites. Une piste alternative pourrait donc consister à effectuer un tri parmi les associations générées afin d'en conserver les éléments réellement informatifs par rapport aux requêtes des utilisateurs et à la base d'exemples initiale. D'autre part, le coût computationnel induit par l'accroissement de la taille de la base pourrait être réduit par simple limitation du temps alloué à chaque requête. En effet, la médiane de la durée par requête est très souvent basse, et ne dépasse pas 16 secondes sur les bases d'exemples générées. Il conviendra au préalable d'étudier la répartition des bonnes réponses données par rapport à la médiane.

Finalement, on peut remarquer le ratio élevé de réponses correctes parmi les réponses proposées par le système. Les résultats obtenus passent rarement sous les 80% de réponses correctes parmi les réponses données. Les exceptions proviennent de la base d'exemples `bash2` où les erreurs proviennent d'une exception mineure à corriger, ainsi que de la base `R` après génération.

Dans le but d'augmenter la couverture du système par rapport à la variété attendue des requêtes de l'utilisateur, et compte tenu de la précision élevée obtenue jusqu'à présent, une approche par relaxation des contraintes peut être envisagée. Il s'agit d'autoriser des approximations dans la vérification et la résolution des analogies pour mieux maîtriser le bruit dans les données et augmenter le rappel du système. Nous présentons cette approche dans le chapitre 6.

Chapitre 6

Relâcher les contraintes analogiques

Sommaire

6.1 Introduction

Comme nous l'avons vu dans les chapitres précédents, le raisonnement par analogie formelle permet à un système apprenant de produire des résultats cohérents à partir d'une base d'exemples de taille limitée. L'utilisation d'une base d'exemples restreinte a comme contrepartie de limiter les variations possibles à partir d'une requête ou d'une commande donnée. Nous avons présenté en 5.5 une méthode basée sur l'analogie afin de générer de la variation à partir d'un corpus restreint. Cette méthode ne permet cependant de générer que des variations formées à partir du vocabulaire de la base. En outre, nous avons vu qu'elle génère également beaucoup de bruit parmi le grand nombre d'exemples produits.

Nous nous intéressons dans ce chapitre à l'adaptation du comportement du système aux situations pour lesquelles aucune réponse exacte ne peut être produite à l'aide du raisonnement analogique. Ces situations, qui ne peuvent être anticipées, proviennent de deux cas de figure :

- La requête comprend au moins un nouveau token qui n'est pas un paramètre
- La requête est une reformulation qui ne peut être obtenue par la génération de nouveaux exemples à partir de la base

Afin d'illustrer ces cas, considérons la base d'exemples simplifiée ci-dessous :

Liste l'ensemble des fichiers du répertoire courant	`ls .`
Liste tous les fichiers du répertoire courant	`ls .`
Liste l'ensemble des fichiers du répertoire tmp	`ls tmp/`
Liste tout ce qu'il y a ici	`ls .`

La requête "*Liste tous les fichiers du dossier courant*" relève du premier cas. Le token "*dossier*" n'est pas présent dans la base et ne peut être ajouté par génération. De plus, il ne correspond pas à un paramètre de la requête. Aucune proportion analogique ne permet donc de produire la commande attendue. Notons qu'à l'inverse, la requête "*Liste tous les fichiers du répertoire actuel*", bien que relevant à première vue du même type de situation, sera traitée par le système à l'aide d'une proportion analogique directe produisant la commande "`ls actuel/`". La validité de la réponse apportée à cette requête ambigüe est déterminée par le contexte, c'est-à-dire l'existence ou non d'un répertoire nommé "actuel" voire même l'intention de l'utilisateur lors de la saisie de sa requête. Nous ne développerons pas ce point ici.

La requête "*Liste l'ensemble des fichiers ici*" relève du second cas de figure. Chacun des tokens est présent dans la base d'exemples, mais aucune proportion directe ni indirecte ne peut être formée avec la requête.

Pour aborder ces deux causes de silence, on peut envisager de rendre le raisonnement analogique plus tolérant. Cela a déjà été proposé par Laurent Miclet et collègues [Miclet *et al.*, 2008] à travers la définition de la notion de dissimilarité analogique appliquée à la résolution d'équations analogiques. Nous proposons ici d'adapter cette résolution pour un lexique de tokens de taille non bornée.

La résolution approchée d'équations analogiques convient essentiellement au cas du raisonnement analogique direct. En effet, la résolution effectuée dans le cas du raisonnement analogique indirect ne fait intervenir que des énoncés en langage formel, les approximations n'y sont donc pas souhaitées. Le raisonnement analogique indirect repose en premier lieu sur la recherche de proportions analogiques valides formées avec des énoncés en langue naturelle uniquement. La méthode de Miclet et collègues peut être appliquée à un quadruplet quelconque grâce à l'algorithme SEQUANA. Cet algorithme est une généralisation de l'algorithme tabulaire de vérification d'analogies (*cf.* section 4.4). À chaque étape, on retient non plus seulement un booléen déterminant si le quadruplet partiel correspondant est une proportion analogique ou non, mais on mémorise sa dissimilarité d'avec la proportion valide la

plus proche. La complexité de cet algorithme est en $\mathcal{O}(|X|.|Y|.|Z|.|T|)$, et son application dans le cas de la résolution indirecte suppose l'énumération exhaustive des quadruplets possibles. Comme nous l'avons vu en section 5.3.1, cette énumération est très coûteuse et requiert l'emploi d'une optimisation (5.3.2). Ces considérations nous conduisent à proposer une notion de déviation de comptage permettant d'utiliser la recherche optimisée de quadruplets fondée sur les arbres de comptages pour trouver des proportions analogiques approchées.

6.2 Résolution analogique approchée

6.2.1 Dissimilarité analogique

Définition

La notion de dissimilarité analogique a été introduite par Laurent Miclet et collègues [Miclet *et al.*, 2008] pour représenter la quantité de modifications qu'il faut appliquer à un quadruplet d'objets pour qu'il soit une proportion analogique valide. Une dissimilarité analogique de 0 signifie donc que le quadruplet est effectivement une proportion analogique, dans le cas contraire, on attend une valeur de dissimilarité strictement positive. Afin de s'appliquer aux quadruplets de manière cohérente, la dissimilarité analogique doit également respecter les propriétés de symétrie des proportions analogiques. La notion proposée par Miclet peut également être considérée comme une distance, car elle respecte l'inégalité triangulaire.

La dissimilarité analogique est définie sur le domaine des quadruplets de séquences comme l'alignement de symboles de coût minimal. Le coût d'un alignement est la somme des dissimilarités analogiques de chaque quadruplet de symboles. Enfin, la dissimilarité analogique d'un quadruplet de symboles peut être définie de manière binaire ou dépendante des symboles eux-mêmes. Dans le premier cas, la dissimilarité analogique est le nombre d'éditions qui séparent le quadruplet de symboles de la proportion analogique la plus proche. Cette proportion au niveau des symboles est nécessairement canonique. L'exemple ci-dessous montre les éditions pour passer du quadruplet de symboles (a, b, ϵ, d) au quadruplet en proportion analogique (a, b, a, b) en deux éditions.

$$\begin{array}{ccccccc} a & & b & & \epsilon & & d \\ a & & b & & a & & d \\ a & : & b & :: & a & : & b \end{array}$$

On peut noter que (a, b, ϵ, d) est également à distance 2 de $(\epsilon, d, \epsilon, d)$ ou encore de (a, d, a, d). Le quadruplet initial a donc une dissimilarité analogique de 2. Il s'agit de la valeur maximale de dissimilarité pour un quadruplet de symboles. En conséquence, la valeur de dissimilarité analogique d'un quadruplet de séquences est comprise entre 0 et $2n$ où n est la longueur en nombre de facteurs de l'alignement de coût minimal. L'exemple ci-dessous montre un alignement de coût minimal pour un quadruplet de séquences de mots. Les dissimilarités analogiques de chaque alignement de symboles sont indiqués dans la dernière ligne du tableau. Le quadruplet de séquences a donc une dissimilarité analogique de 4.

charge	ϵ	le	script	ϵ	`.bashrc`
montre	ϵ	le	fichier	nommé	`f.txt`
charge	ϵ	le	fichier	ϵ	`.profile`
montre	-moi	le	fichier	nommé	`README.md`
0	1	0	1	0	2

La dissimilarité analogique d'un quadruplet de symboles peut également être définie de manière non binaire, on s'appuie alors sur une matrice de dissimilarités, non analogiques cette fois, predéfinie pour l'ensemble des symboles possibles. Pour conserver le même comportement que dans le cas binaire, on s'assure que chaque valeur de la matrice est comprise entre 0 et 1. Cette définition suggère que les symboles ne sont en réalité pas atomiques, et peuvent être représentés par leurs composantes sur lesquelles peut s'appliquer la mesure de dissimilarité. Si les symboles sont des mots, ils sont décomposés en phonèmes, syllabes ou caractères, s'ils sont des caractères, ils peuvent être décomposés en leurs propriétés graphiques telles que la casse, le diacritique, l'accent.

Algorithme SOLVANA

Miclet et collègues [Miclet *et al.*, 2008] proposent un algorithme de résolution approchée d'équations analogiques basé sur la notion de dissimilarité. Il s'appuie sur la procédure de résolution par programmation dynamique, et y ajoute de nouvelles transitions possibles dans le cube de résolution. Étant donné trois séquences X, Y et Z, les transitions utilisées lors de la résolution classique sont :

t_1 : lire un même symbole sur X et sur Y

t_2 : lire un même symbole sur X et sur Z

t_3 : lire un symbole sur Y et l'écrire sur la sortie

t_4 : lire un symbole sur Z et l'écrire sur la sortie

La relaxation consiste ici à considérer les variations possibles autour de ces transitions :

t_5 : lire un symbole sur X

t_6 : lire un symbole sur Y et sur Z

t_7 : lire un symbole sur X, sur Y, et sur Z

De plus, pour chaque transition, tous les symboles du lexique Σ sont considérés comme sorties possibles. Chaque couple transition/émission est associé à un quadruplet de symboles alignés, pour lequel on détermine une valeur de dissimilarité. Le couple associé à la plus faible dissimilarité analogique est sélectionné pour passer à l'itération suivante.

Cette approche, dans sa version naïve, augmente sensiblement la complexité algorithmique par rapport à l'algorithme initial. L'inclusion des transitions supplémentaires ajoute un coefficient $\frac{7}{4}$, et l'énumération de l'ensemble du lexique ajoute un facteur $|\Sigma|$. Cela résulte en une complexité totale en $\mathcal{O}\left(|X|.|Y|.|Z|.|\Sigma|\right)$ qui peut rapidement devenir incontrôlable pour des tailles importantes de lexique.

6.2.2 Application dans un cadre plus général

La méthode de résolution approchée telle que nous venons de la décrire présente une limitation importante liée à la taille du lexique. En effet, un lexique contenant un grand nombre de symboles risque de ralentir significativement la résolution à cause de son énumération. Cet effet est particulièrement problématique lorsque les symboles n'ont pas une longueur fixe en nombre de caractères. Plutôt que les caractères, une segmentation au niveau des mots, par exemple, induit une taille du lexique égale au nombre total de mots dans la base d'exemples. De plus, ce lexique n'est jamais complet et peut (doit) régulièrement être enrichi. Pour employer la résolution approchée dans ce contexte, il est indispensable de minimiser l'influence de la taille du lexique sur la durée d'exécution.

Cela peut être fait en ignorant, pour chaque transition considérée, l'énumération exhaustive du lexique. Le symbole écrit sur la sortie doit donc être sélectionné parmi une liste fixe de possibilités, ne

dépendant pas de la taille du lexique. De plus, nous proposons de limiter la combinatoire induite par les transitions possibles en ignorant les transitions composites, c'est-à-dire les transitions qui peuvent être reproduites par une suite d'autres transitions. Ces transitions sont $t_6 = t_3 \circ t_4 = t_4 \circ t_3$ et $t_7 = t_1 \circ t_4 = t_2 \circ t_3$.

6.3 Recherche approchée de proportions

Dans le cadre de notre application du raisonnement analogique pour le transfert de langage, l'opération de résolution analogique peut intervenir à deux niveaux : lors du transfert direct pour la génération d'une solution à partir d'un triplet mixte $[R_1 : C_1 :: R_{in} :?]$ ou encore lors du transfert indirect pour la génération d'une solution à partir d'un triplet de commandes de la base d'exemples $[C_1 : C_2 :: C_3 :?]$ (*cf.* chapitre 5). Il reste cependant une phase qui n'est pas concernée par la résolution approchée, la recherche de proportions analogiques entre requêtes $[R_1 : R_2 :: R_3 : R_{in}]$ pour le transfert indirect. Or cette opération est tout autant sensible au bruit dans le corpus ou les requêtes soumises au système que le sont la résolution de triplets mixtes et la résolution de triplets de commandes [1].

Miclet [Miclet *et al.*, 2008] a également proposé un algorithme (SEQUANA) permettant de calculer la dissimilarité analogique d'un quadruplet d'objets. La complexité algorithmique de SEQUANA pour un quadruplet de séquences (X, Y, Z, T) est en $O(|X|.|Y|.|Z|.|T|)$, ce qui devient rapidement problématique pour des séquences de plus d'une cinquantaine d'éléments, d'autant que ce polynôme est multiplié par un coefficient constant de 15 [2]. D'autre part, étant donné une requête R en langue naturelle, la recherche du triplet de requêtes de la base d'exemples formant avec R la proportion ayant la plus faible dissimilarité analogique à l'aide de SEQUANA requiert l'énumération de tous les triplets possibles dans la base. Cette opération de complexité cubique n'est pas réaliste car incompatible avec un enrichissement de la base au delà d'un certain ordre de grandeur. Miclet et collègues proposent également l'algorithme FADANA permettant de constituer un quadruplet de dissimilarité analogique minimale plus efficacement.

Nous avons décrit en section 5.3.2 une optimisation de la recherche de proportions analogiques dans la base, fondée sur les arbres de comptage. Cette optimisation s'appuie sur une propriété des proportions analogiques que nous rappelons ici : si un quadruplet de séquences (X, Y, Z, T) est en relation de proportion analogique, alors pour chaque symbole s, la somme des occurrences (ou le comptage) de s dans X et T est égal à la somme de ses occurrences dans Y et Z. Cette propriété ne tient plus si l'on considère non plus seulement les quadruplets proportionnels, mais aussi les quaduplets dont la dissimilarité analogique est supérieure à 0. Cependant, de même que la dissimilarité analogique, le déséquilibre du comptage des tokens dans le quadruplet de séquences augmente à mesure que le quadruplet s'éloigne d'une proportion valide. On définit formellement le **déséquilibre du comptage d'un quadruplet de séquences*** $\delta(X, Y, Z, T)$ comme la valeur absolue de la différence entre le comptage des tokens dans (X, T) et (Y, Z).

$$\delta\,(X, Y, Z, T) = |(|X| + |T|\,) - (\,|Y| + |Z|)|$$

De même, pour le déséquilibre de comptage d'un token particulier t :

$$\delta_t\,(X, Y, Z, T) = |(|X|_t + |T|_t\,) - (\,|Y|_t + |Z|_t)|$$

On a reporté dans le tableau ci-dessous le déséquilibre du comptage pour chaque symbole dans les quadruplets de gauche, le total Σ, ainsi que la dissimilarité analogique DA.

1. On suppose de plus que la résolution de triplets de commandes est moins sujette aux problèmes causés par le bruit, car elle ne fait intervenir que des éléments provenant d'un langage formel.
2. Il s'agit des 15 transitions possibles lorsqu'on considère l'exploration des chemins analogiques approchés

				δ					
X	Y	Z	T	a	b	c	d	Σ	DA
a	a	b	b	0	0	0	0	0	0
a	b	a	c	0	-1	+1	0	2	1
abc	cc	aab	cb	-1	+1	-1	0	3	2
abc	adc	cba	cad	0	0	0	0	0	2

Les valeurs ne sont pas tout à fait corrélées, car les deux mesures de distance ne se comportent pas de la même manière pour les substitutions et les échanges de position. Cependant, un déséquilibre de comptage de à n entrainera toujours une dissimilarité d'au moins $\frac{n}{2}$ correspondant aux étapes minimales pour rétablir le comptage correct, nécessaire pour obtenir une proportion.

Pour guider la recherche approchée basée sur l'arbre de comptage, nous proposons d'utiliser le déséquilibre global du comptage des symboles comme mesure de déviation par rapport à la proportion d'analogie correcte. La déviation est notée Δ et est donnée par la formule suivante :

$$\Delta = \sum_{s \in \mathcal{L}} abs\left(\left(|X|_s + |T|_s\right) - \left(|Y|_s + |Z|_s\right)\right)$$

La recherche par arbre de comptage exacte retient à l'étape i seulement les chemins qui vérifient l'équilibre du comptage pour tous les symboles d'indice inférieur ou égal à i. L'algorithme approché modifie ce dernier pour qu'à l'étape i soient retenus tous les chemins dont le déséquilibre de comptage ne dépasse pas une déviation fixée Δ. L'algorithme 6.1 en propose les détails d'implémentation, il est repris de la recherche par arbre de comptage proposée par Philippe Langlais et François Yvon [Langlais et Yvon, 2008].

L'algorithme approché ne teste plus à chaque itération si le comptage est équilibré, mais s'il est moins déséquilibré que la borne supérieure Δ donnée. Il n'est pas possible de tester directement si le déséquilibre correspond exactement à Δ, car le calcul final dépend également de toutes les étapes ultérieures ($> i$) dont on ne connaît pas le nombre à l'avance. Il en résulte que le calcul de l'ensemble des quadruplets de séquences de déviation Δ permet aussi de produire dans le même temps les résultats analogues pour chaque valeur de déviation comprise entre 0 et Δ.

Comparé à l'algorithme initial, la relaxation contraint à parcourir $2\Delta + 1$ fois plus de chemins ouverts. Le facteur 2 est dû au fait que la déviation concerne autant l'insertion d'un symbole que sa suppression. En pratique cependant, cette exploration est limitée par la forme de l'arbre, c'est-à-dire par la composition de la base. De même que dans le cas de l'algorithme exact, la complexité algorithmique de la recherche approchée par arbre de comptage reste en $\mathcal{O}\left(n^3\right)$ pour une base de taille n. La réduction effective du temps d'exécution est due au fait que le nombre réel de chemins explorés est plus petit que le total, en particulier pour les séquences longues. En effet, ces séquences ont un comptage de symboles spécifique qui est exponentiellement moins partagé avec le reste de la base que la séquence est longue, en supposant que les comptages des séquences de la base respecte la loi de Zipf.

Nous avons vu (5.3.2) que le critère du comptage est suffisamment efficace pour se permettre d'ignorer la phase de vérification du quadruplet. La situation est un peu différente avec la relaxation proposée, puisque comme exposé plus haut, une recherche avec une déviation de n ne donnera pas l'ensemble des quadruplets dont la dissimilarité analogique est de n, ni même $f(n)$. De même que pour notre utilisation de la recherche par arbre de comptage exacte, nous ignorerons donc la vérification analogique des quadruplets sélectionnés. Ce choix est d'autant moins problématique qu'il s'agit déjà de quadruplets relâchés.

Pour identifier l'approximation qui a été faite, il peut être intéressant de calculer, les solutions des équations analogiques $[X, ?, Z, T]$ et $[X, Y, ?, T]$ si elles existent. En effet, on peut considérer les

Algorithme 6.1 : Recherche relâchée de quadruplets à l'aide d'un arbre de comptage

Entrées : Une paire de formes $P_{in} = (f^1_{in}, f^2_{in})$, une déviation maximale Δ et un arbre de comptage T utilisant le lexique $\mathcal{L}$

Sorties : L'ensemble des paires $P_{out} = (f^1_{out}, f^2_{out}, \delta)$ de T telles que $\delta = |count(P_{in}) - count(P_{out})| \leq \Delta$

```
counts ← encode(s¹_in, s²_in) ; frontier ← {(root(T), root(T), 0)} ; i ← 1
tant que i ≤ |L| ∧ frontier ≠ {} faire
    res ← {}
    ν ← counts[L[i]]
    pour tous les (n1, n2, δ) ∈ frontier faire
        pour adjust = 0 → Δ − δ faire
            si n1.index = n2.index = i alors
                pour chaque c1 ∈ n1.c* faire
                    pour chaque c2 ∈ n2.c* faire
                        si |c1.ν + c2.ν − ν| = adj alors
                            res ← res ∪ {(c1, c2, δ + adj)}
                        fin
                    fin
                fin
                pour c1 ∈ {n1.c*[ν − adj], n1.c*[ν + adj]} faire
                    si c1 ≠ nil ∧ |n2.forms| > 0 alors
                        res ← res ∪ {(c1, n2, δ + adj)}
                    fin
                fin
                pour c2 ∈ {n2.c*[ν − adj], n2.c*[ν + adj]} faire
                    si c2 ≠ nil ∧ |n1.forms| > 0 alors
                        res ← res ∪ {(n1, c2, δ + adj)}
                    fin
                fin
            sinon si n1.index = i alors
                pour s ∈ {n1.c*[ν − adj], n1.c*[ν + adj]} faire
                    si s ≠ nil alors
                        res ← res ∪ {(s, n2, δ + adj)}
                    fin
                fin
                si ν = adj ∧ |n1.forms| > 0 alors
                    res ← res ∪ {(n1, n2, δ + adj)}
                fin
            sinon si n2.index = i alors
                pour s ∈ {n2.c*[ν − adj], n2.c*[ν + adj]} faire
                    si s ≠ nil alors
                        res ← res ∪ {(n1, s, δ + adj)}
                    fin
                fin
                si ν = adj ∧ |n2.forms| > 0 alors
                    res ← res ∪ {(n1, n2, δ + adj)}
                fin
            sinon si ν = adj alors
                res ← res ∪ {(n1, n2, δ + adj)}
            fin
        fin
    fin
    frontier ← res
    i ← i + 1
fin
retourner {(f1, f2, δ) : f1 ∈ p1.forms, f2 ∈ p2.forms, (p1, p2, δ) ∈ frontier}
```

couples (Y, Y') et (Z, Z') comme des justifications de la réponse finale du système, Y' et Z' étant les solutions respectives des deux équations.

6.4 Confrontation aux données de la tâche

Cette section est dédiée à la description de l'architecture du système mis en place, et des ensembles de test utilisés. Nous détaillons ensuite les résultats obtenus avec et sans utilisation de la déviation.

6.4.1 Mise en place expérimentale

Les expériences présentées ici sont fondées sur une version mise à jour du système présenté dans le chapitre 5. Le système utilisé ici s'en distingue par l'utilisation de l'algorithme de programmation dynamique pour la phase de résolution décrite en section 4, au lieu de l'algorithme par échantillonnage décrit au même endroit. Les résultats n'ont pas subi de changement notable suite à cette mise à jour. La durée d'exécution, en revanche, s'est trouvée modérément améliorée car l'algorithme par programmation dynamique ne requiert pas de calculer explicitement les résultats dont le nombre de facteurs associés n'est pas minimal (*cf.* la description des algorithmes section 4).

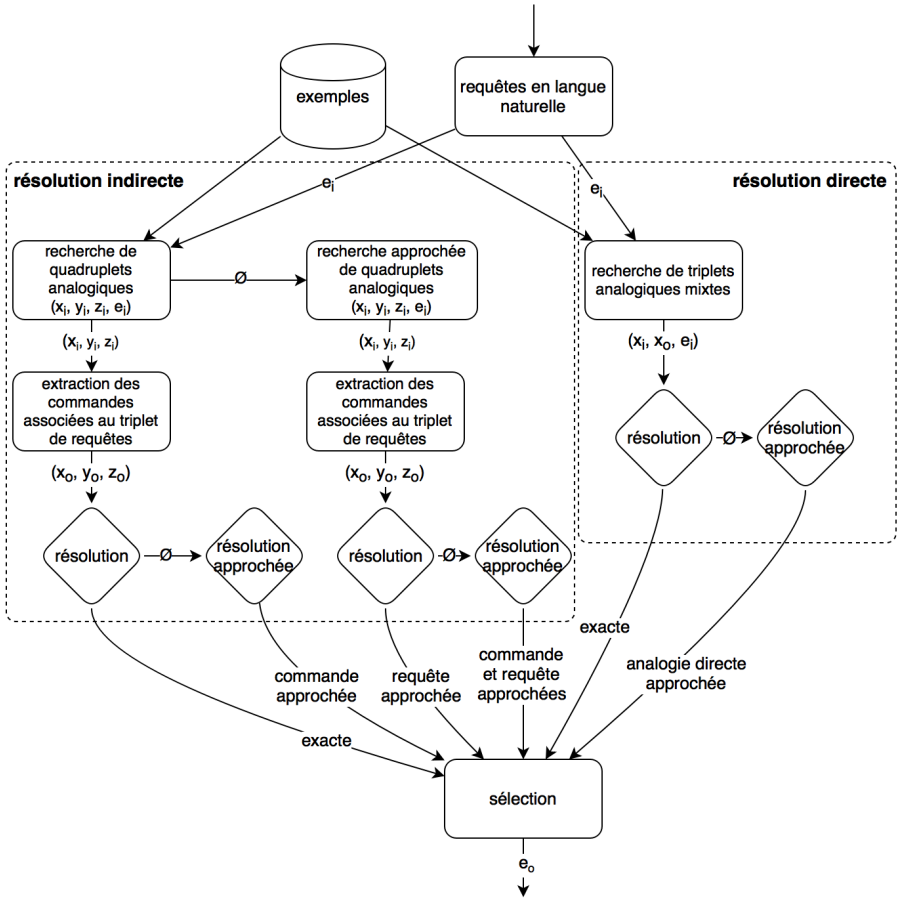


FIGURE 6.1 – Aperçu de l'architecture des algorithmes de résolution

L'architecture générale du système est réprésentée sur la figure 6.1. L'entrée est composée de requêtes en langue naturelle, et la sortie de commandes en langage formel. On retrouve les procédures

de résolution selon les stratégies directe et indirecte et les procédures de recherche dans la base présentées précédemment. La figure comporte également la procédure de recherche approchée dans la base ainsi que celle de résolution approchée d'équations analogiques. Ces procédures relâchées ne sont appelées que lorsque la déviation maximale autorisée Δ est supérieure à 0, et dans les cas où aucune solution n'a pu être trouvée par les procédures exactes. Elles sont en effet plus coûteuses en temps que les procédures exactes. En outre, on admet que les résultats obtenus de manière exacte sont toujours préférables à des résultats obtenus de manière approchée. Autrement dit, on suppose que pour tous les cas où un résultat exact et un résultat approché peuvent être proposés, si au moins l'un des deux est correct, alors le résultat exact l'est.

Comme décrit dans les sections précédentes, la notion de déviation concernant l'opération de recherche est différente de la notion de déviation concernant la résolution, qui fait alors référence à la dissimilarité analogique au sens de Miclet. Dans la suite, on différencie la déviation maximale autorisée selon qu'elle s'applique à la recherche relâchée (Δ_{rchr}) ou à la résolution relâchée (Δ_{res}).

Les sections suivantes présentent respectivement les résultats obtenus pour ces expériences en utilisant la base `bash1`, `R` et `bash2` (*cf.* section 2.4).

6.4.2 Résultats sur la base `bash1`

Lors de l'expérience utilisant la base BASH1, les ensembles de tests sont les mêmes que pour l'expérience présentée dans le chapitre 5 : un ensemble constitué d'une partie extraite de la base de connaissances, appelé QUOTIDIEN, et un ensemble collecté séparément avec variation systématique des paramètres, appelé NOUVEAU. Chacun d'eux contient 77 requêtes à transférer.

Les tests ont été conduits en faisant varier la déviation autorisée pour la recherche et pour la résolution sur chacun des ensembles de test. Les méthodes de résolution directe et indirecte ont été utilisées ensemble.

Les tableaux 6.1 et 6.2 montrent les scores et la durée d'exécution obtenus respectivement pour l'ensemble QUOTIDIEN et pour l'ensemble NOUVEAU, selon la variation du paramètre Δ_{rchr}. De haut en bas, les lignes correspondent : au nombre de réponses données par le système, au nombre de réponses correctes parmi celles-ci, aux valeurs de précision et de rappel associées, à la durée moyenne d'exécution en secondes pour une recherche exhaustive, et enfin à la durée moyenne d'exécution en arrêtant la recherche dès la première réponse. Comme attendu, le nombre de réponses augmente progressivement en fonction de la déviation autorisée, jusqu'à 64 sur 77 sur l'ensemble QUOTIDIEN pour $\Delta_{rchr} = 5$. Le même comportement est observé sur l'ensemble NOUVEAU, avec une moindre amplitude cependant (35 réponses sur 77 à $\Delta_{rchr} = 5$). Le nombre de réponses correctes augmente également à un rythme moins soutenu, mais stagne rapidement sur l'ensemble NOUVEAU, ce qui résulte en une forte diminution de la précision. La durée d'exécution augmente également beaucoup dès que Δ_{rchr} dépasse 2.

Les tests équivalents pour une variation de Δ_{res} sont beaucoup moins coûteux en temps, comme le montrent les tableaux 6.3 et 6.4, mais ne permettent qu'une amélioration limitée du nombre total de réponses correctes. Notons que les scores obtenus avec une déviation Δ_{rchr} ne sont pas tout à fait comparables aux scores obtenus avec une déviation Δ_{res}.

Les résultats obtenus étaient en partie attendus, mais montrent des spécificités intéressantes entre la déviation au niveau de la recherche seulement et au niveau de la résolution seulement.

6.4.3 Résultats sur la base `R`

De même que pour l'expérience du chapitre 5, un seul ensemble de test a été utilisé pour la base `R`, ne disposant pas d'un ensemble collecté auprès de participants indépendants.

Δ_{rchr}	0	1	2	3	4	5
réponses	36	43	50	53	59	64
correctes	35	42	49	51	57	61
précision	0.97	0.98	0.98	0.96	0.97	0.95
ratio de réponses données	0.47	0.56	0.65	0.69	0.77	0.83
temps par requête moyen (exhaustif)	0.4	2.4	14.1	35.5	82.1	163.8
temps par requête moyen	0.3	1.3	5.0	13.7	32.9	67.0

TABLEAU 6.1 – Résultats pour une déviation sur la recherche sur l'ensemble QUOTIDIEN

Δ_{rchr}	0	1	2	3	4	5
réponses	9	10	14	18	27	35
correctes	9	10	10	11	11	11
précision	1	1	0.71	0.61	0.41	0.31
ratio de réponses données	0.12	0.13	0.18	0.23	0.35	0.45
temps par requête moyen (exhaustif)	0.0	0.5	3.1	10.5	27.0	56.4
temps pas requête	0.0	0.3	2.2	8.1	20.2	42.1

TABLEAU 6.2 – Résultats pour une déviation sur la recherche sur l'ensemble NOUVEAU

Δ_{res}	0	1	2	3	4	5
réponses	36	48	55	61	66	71
correctes	35	41	42	42	42	42
précision	0.97	0.85	0.76	0.69	0.63	0.59
ratio de réponses données	0.47	0.62	0.71	0.79	0.85	0.92
temps par requête moyen (exhaustif)	0.4	0.9	1.3	1.8	2.2	2.7
temps par requête moyen	0.3	0.4	0.5	0.5	0.6	0.6

TABLEAU 6.3 – Résultats pour une déviation sur la résolution sur l'ensemble QUOTIDIEN

Δ_{res}	0	1	2	3	4	5
réponses	9	20	30	39	44	52
correctes	9	10	12	12	12	12
précision	1	0.50	0.40	0.31	0.27	0.23
ratio de réponses données	0.12	0.26	0.39	0.51	0.57	0.68
temps par requête moyen (exhaustif)	0.4	0.9	1.3	1.8	2.2	2.7
temps par requête moyen	0.3	0.4	0.5	0.5	0.6	0.6

TABLEAU 6.4 – Résultats pour une déviation sur la résolution sur l'ensemble NOUVEAU

Les tableaux 6.5 et 6.6 donnent respectivement les scores obtenus avec une déviation au niveau de la recherche et une déviation au niveau de la résolution analogique. On retrouve la même tendance qu'observée précédemment pour `bash1` et qui était attendue : baisse de la précision et augmentation du ratio de réponses données. Le nombre absolu de réponses correctes données par le système augmente assez régulièrement avec la déviation sur la recherche, mais très difficilement avec la déviation à la résolution.

Δ_{rchr}	0	1	2	3	4	5
réponses	22	26	37	40	53	59
correctes	21	23	25	27	32	32
précision	0.95	0.88	0.68	0.68	0.60	0.54
ratio de réponses données	0.29	0.34	0.48	0.52	0.69	0.77
temps par requête moyen (exhaustif)	0.5	3.3	13	42	105	242

TABLEAU 6.5 – Résultats pour une déviation sur la recherche sur la base `R`

Δ_{res}	0	1	2	3	4	5
réponses	22	36	43	58	65	69
correctes	21	22	22	21	21	23
précision	0.95	0.61	0.51	0.36	0.32	0.33
ratio de réponses données	0.29	0.47	0.56	0.75	0.84	0.90
temps par requête moyen (exhaustif)	0.5	1.0	1.6	2.2	3.0	3.9

TABLEAU 6.6 – Résultats pour une déviation sur la résolution sur la base `R`

On notera des fluctuations observables dans le nombre de réponses correctes données par le système. Elles sont dues à la multiplicité des réponses possibles en autorisant une déviation donnée. Le système ne dispose pas d'heuristique pour sélectionner une réponse parmi celles-ci, un choix aléatoire est donc effectué et conduit à cette fluctuation lorsque toutes les options ne sont pas homogènement correctes ou incorrectes. L'exemple 6.1 montre l'effet de cette sélection aléatoire sur le traitement de deux requêtes, et causant les fluctuations constatées dans le tableau 6.6.

```
met                DIGITS  à  20
options  (         DIGITS  =  20  )
option             warn    à  2
options  (         warn    =  2   )

---

         met       DIGITS  à  20
options  (         DIGITS  =  20  )
         option    warn    à  2
options  option    warn    =  2   )
```

EXEMPLE 6.1 – Deux proportions analogiques alignées, valides à une déviation de 1 près. Les tokens en rouge sont ceux sur lesquels porte la déviation. L'avant dernière ligne de chaque proportion correspond à la requête soumise, et juste en dessous de chacune se trouve la réponse du système.

met DIGITS à 20	:	**option** warn à 2
↕	::	↕
`options ( DIGITS = 20 )`	:	`options ( warn = 2 )`

EXEMPLE 6.2

met DIGITS à 20	:	option warn à 2
↕	::	↕
`options ( DIGITS = 20 )`	:	`options option warn = 2 )`

EXEMPLE 6.3

La durée d'exécution est ici aussi beaucoup plus affectée par l'ajout de déviation lors de la recherche que lors de la résolution. Pour une déviation sur la recherche, le tableau 6.5 montre que le temps de réponse par requête est en moyenne multiplié par $3,4$ par unité de dévation. En revanche, la durée croît beaucoup plus modérément en ce qui concerne la déviation sur la résolution analogique, passant de $0,5$ seconde dans déviation à $3,9$ secondes par requête pour une déviation de 5 mots.

6.4.4 Résultats sur la base `bash2`

La base d'exemples `bash2` a également donné lieu à une évaluation sur un seul ensemble de test. Comme précisé en section 5.7.2, les conditions de sa collecte rendent l'évaluation sur un sous-ensemble extrait de la base comparable à l'évaluation sur l'ensemble NOUVEAU pour la base `bash1` par rapport à la diversité des commandes et des formulations, mais également comparable à l'évaluation sur QUOTIDIEN de `bash1` lorsque les commandes concernées sont plus fréquentes dans les historiques des participants à la collecte.

Δ_{rchr}	0	1	2	3	4	5
réponses	24	25	31	36	43	54
correctes	19	18	23	26	29	29
précision	0.79	0.72	0.74	0.72	0.67	0.54
ratio de réponses données	0.31	0.32	0.40	0.47	0.56	0.70
temps par requête moyen (exhaustif)	0.4	4.0	24	132	597	1842

TABLEAU 6.7 – Résultats pour une déviation sur la recherche sur la base `bash2`

Δ_{res}	0	1	2	3	4	5
réponses	24	32	39	52	55	60
correctes	19	19	18	19	20	19
précision	0.79	0.59	0.46	0.37	0.36	0.32
ratio de réponses données	0.31	0.41	0.51	0.68	0.71	0.78
temps par requête moyen (exhaustif)	0.4	0.5	0.8	1.0	1.4	1.8

TABLEAU 6.8 – Résultats pour une déviation sur la résolution sur la base `bash2`

On retrouve des évolutions en fonction de la déviation similaires à ce qui a été observé jusqu'à présent. Les tableaux 6.7 et 6.8 montrent une augmentation claire du nombre de bonnes réponses, passant de 25% à 38%, avec la déviation sur la recherche, mais une stagnation avec déviation sur la

résolution. De même, on constate un ralentissement conséquent du traitement pour la déviation sur la recherche, alors que la déviation sur la résolution n'occasionne qu'une augmentation 1000 fois moindre du délai à déviation 5.

6.5 Conclusion

6.5.1 Discussion

Analyse des résultats

La relaxation de la contrainte de comptage pour la recherche de requêtes dans la base d'exemples permet une augmentation remarquable du rappel tout en limitant l'impact sur la précision sur l'ensemble QUOTIDIEN. Le facteur le plus limitant est en réalité la durée d'exécution qui devient rapidement irréaliste pour une utilisation instantanée. C'est également un facteur à surveiller lors de l'utilisation de base d'exemples de plus grande taille ou encore pour la mise en place d'un enrichissement incrémental de la base. Au delà d'une déviation (sur la recherche) de 2, ou pour des bases plus grandes, il devient nécessaire d'utiliser des stratégies de sélection efficaces pour conserver des délais de réponse raisonnables.

Les tests faisant varier Δ_{rchr} sur l'ensemble NOUVEAU ne montrent en revanche pas une amélioration aussi spectaculaire. C'est un résultat cohérent puisque l'essentiel des réponses correctes sur cet ensemble de test sont produites grâce à des analogies directes (voir l'étude des analogies directes et indirectes présentée en section 5.6). Une faible augmentation du nombre de réponses correctes montre malgré tout que certaines requêtes de l'ensemble NOUVEAU sont suffisamment proches de la base d'exemples disponible pour être capturées après relâchement de la contrainte de comptage. Cependant, la réponse avec succès à des requêtes proches n'est possible que lorsque la commande à produire ne contient aucun symbole absent de la base d'exemples. En effet, il n'est pas pertinent de générer un symbole inconnu, car on ne peut définir une stratégie cohérente pour préférer *a priori* la génération d'un symbole plutôt que d'un autre parmi l'infinité des possibilités.

Concernant les tests faisant varier Δ_{res}, les réponses correctes supplémentaires ne sont obtenues qu'au prix d'une chute importante de la précision. On peut en déduire que le bruit le plus critique dans la base d'exemples se trouve dans la partie en langue naturelle. Ce n'est pas très surprenant, mais il est alors intéressant de remarquer que la relaxation de la résolution a également permis à certaines équations irrésolues de donner des solutions. L'utilisation de la résolution approchée reste en pratique nettement moins coûteuse en temps que la relaxation du comptage car sa complexité algorithmique ne dépend pas de la taille de la base d'exemples.

Finalement et malgré la taille modeste des données de test, ces résultats montrent clairement que les méthodes de transfert de langage par analogie formelle ont une importante marge de progression concernant le contrôle du bruit dans la base d'exemples.

Déviation et dissimilarité

La modification proposée de l'algorithme de résolution d'équations analogiques par programmation dynamique suit une motivation différente de celle proposée par Miclet et collègues. Leur approche est utilisée pour la production de solutions déviées afin de produire une réponse aussi souvent que possible lorsque la déviation n'est pas trop élevée. Cependant, l'énumération de l'ensemble des symboles possibles pour considérer les insertions possibles n'est pas réalisable avec des symboles de taille variable. C'est notre cas puisque l'unité de traitement utilisée ici est le mot.

Dans le cas de Miclet, les valeurs de dissimilarité analogique sont définies entre caractères par un ensemble prédéfini de caractéristiques sur l'alphabet. C'est un point intéressant à explorer car il est

appliquable aux mots et permet de restreindre les candidats à l'insertion. Il requiert malgré tout un parcours exhaustif des symboles possibles.

6.5.2 Bilan

Nous avons présenté dans ce chapitre des propositions pour réduire l'hypersensibilité des méthodes analogiques au bruit de la base de connaissances. Nous avons proposé une relaxation de l'algorithme de résolution fondé sur [Miclet *et al.*, 2008], et une relaxation des contraintes de la recherche optimisée d'équations analogiques dans la base d'exemples à partir de [Langlais et Yvon, 2008]. L'hypothèse était que la relaxation des contraintes fortes inhérentes aux analogies formelles pourrait permettre de trouver de nouvelles solutions jusqu'ici inaccessibles. Bien que la relaxation de la résolution seule produise des scores plutôt moyens, la recherche par arbre de comptage relâchée, malgré son coût computationnel non négligeable, a montré une amélioration encourageante du nombre de réponses correctes.

Les résultats produits par ce système peuvent être améliorés grâce à la modalité dialogique. Un moyen d'éviter les problèmes posés par les commandes incorrectes en contexte interactif est de simplement demander confirmation avant exécution de ladite commande. Cela peut être fait, comme pour tout système, lors de la suggestion de la commande, mais cette solution ne convient que lorsque l'utilisateur est compétent pour juger de la validité de la commande. La recherche approchée indirecte permet une alternative intéressante : lorsqu'un quadruplet a été constitué sur le domaine des requêtes en langue naturelle avec une déviation supérieure à 0, on peut utiliser trois des quatre éléments (par exemple X, Y et Z) [3] pour calculer une solution exacte alternative T'. Le système peut alors demander à l'utilisateur s'il estime que $T = T'$. Si c'est le cas, alors l'approximation est justifiée et la commande correspondante peut être exécutée, sinon la recherche peut se poursuivre. Cette interaction peut également se révéler pertinente lorsque plusieurs solutions approchées différentes sont trouvées, afin d'en identifier une correcte. Ce faisant, le système peut construire incrémentalement une base de synonymie des requêtes.

Les pistes ouvertes à l'issue des propositions exposées dans ce chapitre sont multiples. Tout d'abord, les résultats obtenus en termes de durée d'exécution renforcent l'importance de travailler à la réduction de la complexité des algorithmes. Cela peut être envisagé par une optimisation des algorithmes existants, l'ajout d'heuristiques pour le parcours de la base ou la résolution d'analogies, ou bien la mise au point d'une méthode de compression de la base de connaissances afin de réduire sa taille tout en conservant l'essentiel de l'information contenue. Il s'agit semble-t-il d'un point saillant dans le problème du transfert de langage par analogie, car les résultats présentés plus haut montrent que de nouvelles réponses correctes sont trouvées même pour des valeurs élevées de déviation. Une autre piste concerne l'utilisation d'une unité de segmentation variable potentiellement plus grande que le mot. Cela peut être utile notamment pour appliquer la déviation sur des expressions multi-mots qui ne sont actuellement pas gérées correctement. D'autres types de déviations pourraient également être intéressantes dans le but d'éviter la suppression ou le remplacement de symboles cruciaux comme par exemples les paramètres des requêtes.

3. La résolution peut être tentée pour tous les triplets possibles parmi le quadruplet formé. Cependant l'utilisateur comprendra probablement mieux le raisonnement du système s'il est lié à sa propre requête T. À noter que tous les triplets ne produiront pas nécessairement de solution.

Chapitre 7

Vers un apprentissage incrémental

Sommaire

7.1 Introduction

Nous étudions dans ce chapitre le comportement du système dans le cadre de l'apprentissage incrémental, ou apprentissage séquentiel, par comparaison à l'apprentissage hors ligne (batch learning). À cette fin, nous proposons plusieurs expériences de simulation d'un apprentissage incrémental sur le modèle proposé au chapitre 1. Cette simulation est effectuée grâce à la soumission au système d'une séquence de requêtes. Les commandes attendues pour chacune sont ajoutées incrémentalement à la base d'exemples, de même qu'il en serait par l'interaction avec un utilisateur humain.

L'attrait principal de l'apprentissage incrémental est la capacité d'augmenter toujours plus la précision et/ou la couverture du système. Dans une situation triviale dans laquelle l'ensemble des exemples à connaître est fini, la précision et la couverture atteindraient 1 au moment où le dernier exemple de l'ensemble serait ajouté à la base. Dans la situation réelle d'un nombre d'exemples non borné, l'apprentissage "par cœur" ne suffit pas. La question qui en résulte est donc : dans quelle mesure le système est-il capable d'exploiter chaque nouvel exemple pour augmenter précision et couverture ? Cette question repose fortement sur les propriétés du nouvel exemple par rapport à la composition de la base. Ces dernières sont susceptibles de varier considérablement entre le cas dans lequel toute l'information du nouvel exemple est déjà présente dans la base, et celui dans lequel le nouvel exemple est entièrement inédit. On peut prévoir l'évolution théorique du système de transfert analogique étant donné un nouvel exemple dont on connaît l'apport en termes d'information vis-à-vis de la base existante. Cette évolution théorique peut-être mesurée en nombre de requêtes supplémentaires couvertes ou traitées avec succès par le système après ajout de l'exemple. Cependant, nous ne pouvons pas considérer connaître à l'avance les nouveaux exemples apportés au système en condition réelle, car les besoins de l'utilisateur et son usage de la langue naturelle sont tous deux des domaines ouverts. Nous aborderons donc la question de l'exploitation des nouveaux exemples par le système sous l'angle empirique.

Une tâche d'apprentissage est dite incrémentale si *les exemples d'apprentissage utilisés pour la résoudre deviennent disponibles au cours du temps, habituellement un par un*, ainsi que le définit [Giraud-Carrier, 2000]. L'auteur met en évidence que certaines catégories de tâches, telles que la conception d'agents intelligents, sont naturellement propices à l'emploi de l'apprentissage incrémental. Bien qu'il suscite un intérêt croissant, c'est encore un paradigme relativement marginal parmi les approches de l'apprentissage artificiel. Cependant, quelle que soit la quantité de données utilisées pour l'entraînement d'un système apprenant, une mise à jour de l'algorithme d'apprentissage (en dehors d'un simple ajustement de paramètres) requerra des données également mises à jour. Cela signifie la constitution d'un corpus contenant les propriétés devenues nécessaires. Cet aspect est particulièrement problématique dans le domaine des agents intelligents, où l'interaction avec l'humain contraint la quantité des données collectées. Sous cette perspective, l'apprentissage incrémental est plus réaliste que l'apprentissage hors ligne, car il n'implique pas l'hypothèse du monde clos.

L'incrémentalité soulève la question de l'ordre d'apprentissage des exemples. Même si le score final est similaire, des ordonnancements différents font varier les performances intermédiaires du système. Le comportement humain est difficilement prédictible, il a ici une influence directe sur l'ordonnancement. Dans le domaine de l'enseignement algorithmique (par un humain à une machine), [Cakmak et Thomaz, 2014, Balbach et Zeugmann, 2009] tentent de guider l'enseignement grâce à un ensemble d'instructions prédéfini. L'objectif est d'amener l'humain à proposer des exemples utiles pour le système, qu'il ne proposerait pas naturellement lors d'un enseignement humain-humain. Dans le contexte du raisonnement par analogie, ce guidage ne semble pas nécessaire *a priori*. En effet, un ordonnancement naturel devrait fournir des exemples aux moments où ils sont nécessaires, et l'information inédite de ces exemples est immédiatement exploitée grâce au raisonnement analogique. De plus, notre système interagit davantage avec un utilisateur qu'avec un enseignant. Bien que celui-ci

soit amené à jouer ces deux rôles, il est préférable de limiter les contraintes imposées à l'utilisateur concernant l'enseignement d'exemples au système.

Le raisonnement analogique formel n'a pas été utilisé dans le contexte de l'apprentissage incrémental jusqu'à présent. Cependant, ses propriétés le rendent particulièrement adapté à l'incrément :

- l'absence de compilation couteuse des connaissances de la base d'exemples permet que chaque nouvel exemple soit intégré en temps constant,
- les variations dans les occurrences d'exemples n'influencent pas la génération de la réponse, sa sélection elle, peut être influencée.

7.2 Protocole expérimental

Afin d'évaluer le comportement du système de raisonnement par analogie en condition incrémentale, le protocole le plus direct correspond à mettre en place une interaction suivie avec un utilisateur, en mesurant sur la durée le taux de bonnes réponses du système. Il induit également un biais lié à l'habituation de l'utilisateur au système au cours des échanges. Il est donc possible que l'augmentation du taux de bonnes réponses ne soit pas seulement lié à l'accroissement du nombre d'exemples, mais aussi à une adaptation des requêtes de l'utilisateur. D'autres biais sont également à considérer, comme l'influence des intervalles entre chaque interaction, ou encore les erreurs possibles de l'utilisateur dues à une incompréhension ou à une méconnaissance.

Bien que la confrontation à ces biais rendent l'évaluation plus représentative de la réalité, il est difficile de tous les contrôler pour distinguer leur influence de celle des caractéristiques du système pour l'analyse des résultats. Une approche expérimentale alternative consiste à mettre en place une simulation de l'utilisateur dont on peut définir les propriétés telles que la connaisance du domaine, l'ordre des requêtes ou l'évolution de leur formulation.

Nous avons dressé une liste de conditions expérimentales pertinentes pour l'utilisation du système et pour mettre en évidence l'influence de l'apprentissage incrémental. Il s'agit de : l'ordre d'apprentissage des exemples, le nombre d'exemples appris, le changement d'utilisateur et l'utilisation de la génération analogique d'exemples. Les protocoles de test correspondants avec les résultats d'expériences obtenus sont décrits dans les sous-sections qui suivent.

La base d'exemples utilisée pour cette expérience est `bash1` (voir la section 2.4.2 pour une description détaillée), des comparaisons avec les autres bases collectées sont présentées par la suite. Le tableau 7.1 reprend les chiffres de la base et présente les nombres d'exemples supplémentaires générés par analogie [1]. La base initiale représente un ensemble d'exemples qui a pu être accumulé auprès d'un groupe d'utilisateurs réguliers, on y réfèrera sous l'expression "base QUOTIDIEN".

Elle a été collectée auprès d'un petit groupe de participants en leur remettant la liste des vingt commandes ; la consigne était, pour chaque commande, de rédiger plusieurs formulations de requêtes pour demander naturellement à une personne d'effectuer la commande. Aucune contrainte n'a été donnée concernant ces formulations afin de refléter le plus possible une interaction naturelle. La plupart sont naturellement à l'impératif mais beaucoup aussi sont sous forme interrogative. Quelques unes comprennent des erreurs grammaticales. Les exemples ci-dessous illustrent la forme des associations recueillies :

Crée une copie de foo appelée bar	→	`cp foo bar`
Va dans dir et récupère les derniers commits	→	`cd dir; git pull`
Dans foo, peux-tu me dire combien y a de mots ?	→	`wc -l foo`

1. Les totaux diffèrent de ceux donnés dans la section 5.5 car nous avons ici utilisé l'intégralité de la base pour l'extension analogique au lieu de seulement 435 associations.

	associations	commandes	cohortes
QUOTIDIEN	512	21	20
génération	54243	48	-
NOUVEAU	77	20	20
génération	1205	23	-
totaux génération	73628	230	-

TABLEAU 7.1 – Composition de la base `bash1` utilisée et de son extension par analogie
Les nombres sous *associations* donnent le nombre de paires requête-commande de la base. Les lignes *génération* de chaque base donnent le nombre d'éléments uniques et nouveaux. Le comptage des commandes tient compte de toute variation pour différencier deux commandes.

La seconde base contient un nombre plus restreint d'exemples que l'on peut attribuer à un nouvel utilisateur du système, susceptible de s'exprimer différemment de ce qui a été appris dans la base QUOTIDIEN. Cette seconde base, nommée "base NOUVEAU" dans la suite, a effectivement été collectée auprès d'un volontaire indépendant de la première. La liste de commandes était identique, mais la consigne comprenait l'instruction supplémentaire de faire varier systématiquement les paramètres des commandes (noms de fichiers, nombre d'éléments, chemins d'accès...). De plus, contrairement à la première collecte, aucun exemple d'association n'a été fourni afin de limiter autant que possible l'influence sur les formulations du participant. Les associations qu'elle contient concernent cependant le même ensemble de commandes. Le tableau décrit également le nombre d'exemples générés pour chacune des deux bases par la méthode présentée précédemment, en tentant de résoudre les équations analogiques formées par chaque triplet constitué à partir de la base.

La base QUOTIDIEN a été disposée selon différents ordonnancement dans les tests :

- groupée par cohortes,
- groupée par cohortes en ordre inverse,
- mélangée.

Une **cohorte**[*] est définie par l'ensemble des exemples qui concernent le même type de commande. Par exemple "*Efface le fichier foo.odt*" (`rm foo.odt`) et "*Supprime récursivement le répertoire bar/*" (`rm -R bar/`) appartiennent à la même cohorte, alors que "*Efface la variable $VAR*" (`unset $VAR`) appartient à une cohorte distincte des précédents exemples.

Par souci de lisibilité des graphes présentés dans la suite, la performance de l'apprentissage est représentée par le nombre *cumulé* de bonnes réponses fournies pendant la progression incrémentale. Les caractéristiques intéressantes à relever sont donc la valeur finale ainsi que la forme des courbes. L'aire sous les courbes n'est pas ici une propriété pertinente pour l'interprétation. Un apprentissage idéal produira une courbe ayant une dérivée tendant rapidement vers 1. La droite d'équation $y = x$ a été ajoutée pour comparaison, elle représente la limite supérieure stricte pouvant être atteinte par un système apprenant incrémentalement. En effet, elle correspond à un système donnant exactement une bonne réponse pour chacune des requêtes qui lui sont soumises. En d'autre termes, il s'agit d'un système qui possède déjà toutes les connaissances nécessaires pour répondre à chaque requête, ce qui est donc impossible en commançant avec une base d'exemples vide.

7.3 Influence de l'ordre d'apprentissage

Dans le but d'analyser l'effet de l'ordonnancement des exemples sur la performance, le système a été testé sur la base QUOTIDIEN de manière incrémentale. Pour chaque itération i, la i-ème requête

est soumise au système, la réponse comparée à la commande attendue, puis la requête associée à sa commande est ajoutée à la base d'exemples du système. Cet ajout systématique peut être vu en contexte interactif comme un processus de validation : le système demande à l'utilisateur si sa proposition est correcte. Si c'est le cas, l'exemple est enregistré, sinon le système demande à l'utilisateur de fournir la commande correcte qui aurait dû être proposée et l'association est ajoutée.

La figure 7.1 montre les performances du système selon l'ordre des requêtes d'entrée. On remarque que sur les 512 requêtes de la base, le système produit en condition incrémentale un peu plus de 50% de réponses correctes, bien que le test sur la base mélangée ait environ 15 réponses de retard sur les autres. Malgré la proximité des scores finaux, la proportion de requêtes correctement répondues commune entre les trois tests ne dépasse pas 48%. En effet, la première requête d'une cohorte à être soumise au système ne peut jamais avoir de réponse correcte, et celle-ci n'est jamais la même pour les trois ordonnancements. La stabilité des résultats peut s'expliquer *a priori* par la propriété de symétrie de l'analogie : si le système est capable de répondre correctement à une requête D ayant vu A, B et C, alors il est également capable de répondre correctement à A étant donné B, C et D. Cette symétrie ne s'applique pas à toutes les permutations mais suit la règle ci-dessous, applicable transitivement :

$$[A : B :: C : D] \Leftrightarrow [A : C :: B : D] \Leftrightarrow [C : A :: D : B]$$

Cependant, cette symétrie ne rend les résultats finaux similaires qu'en supposant que toutes les proportions de la base sont disjointes or ce n'est pas le cas général, comme discuté plus loin.

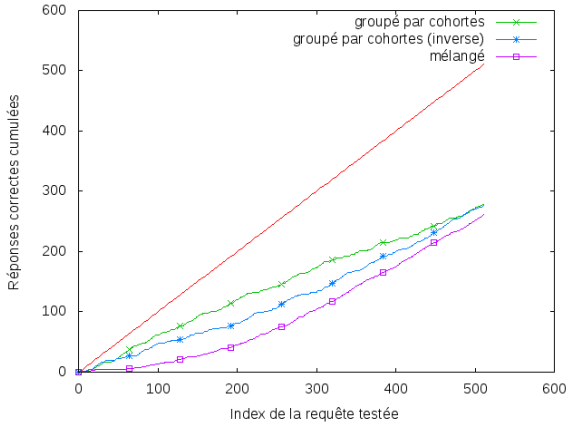


FIGURE 7.1 – Effet de l'ordre des exemples sur les performances de l'apprentissage incrémental sur la base `bash1`

La figure 7.1 montre que l'ordonnancement des exemples a un effet sur la performance. Grouper les exemples donne un meilleur score final et donne des réponses correctes plus régulièrement qu'utiliser la base mélangée. Ce dernier aspect s'explique naturellement par le fait que l'ensemble des connaissances disponibles pour chaque cohorte est acquise séquentiellement, ce qui conduit à produire les bonne réponses qu'elles permettent de trouver immédiatement à l'issue de cette phase locale d'apprentissage. Néanmoins, la courbe donnée par la base mélangée est plus lissée et sa dérivée est toujours croissante. Ce sont des propriétés importantes pour la poursuite de l'apprentissage au delà de ces 512 exemples. Les coefficients directeurs moyens de chacune des courbes sur les 200 dernières requêtes sont, dans l'ordre de la figure, $0,47$, $0,68$ et $0,74$. Si l'exécution se poursuit sur de nouvelles requêtes avec les mêmes taux de réponse, on s'attend à voir la courbe de l'ordonnancement aléatoire dépasser les deux premières. Nous allons à présent analyser de manière plus détaillée les effets de l'ordonnancement en faisant varier le nombre d'exemples appris.

7.4 Arrêt subit de l'apprentissage

Le comportement du système montré au cours des tests avec différents ordonnancements peut aussi être influencé par le nombre d'exemples soumis au total. Même en condition d'apprentissage incrémental, il pourrait être nécessaire d'interrompre l'apprentissage pendant une certaine période, par exemple pour interagir avec un utilisateur final qui n'a pas les connaissances lui permettant d'enseigner les bonnes réponses au système quand une erreur est commise. Nous avons évalué la performance du système en interrompant artificiellement l'incrément de la base à partir de différents états de la base. Les résultats sont donnés par les figures 7.2 et 7.3. Elles présentent dix tests différents effectués avec un arrêt de l'incrément aux indices 50, 100, 150 ... jusqu'à 500, ainsi que le test initial (courbe colorée/ponctuée) sans interruption. On peut également assimiler l'arrêt de l'incrément à l'indice n comme une simulation de l'apprentissage hors ligne avec une base initiale composée des $n-1$ premiers exemples.

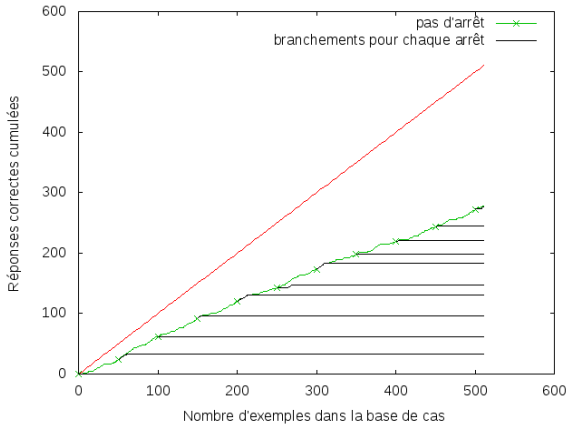


FIGURE 7.2 – Comportement après interruption de l'apprentissage pour une base groupée par cohortes

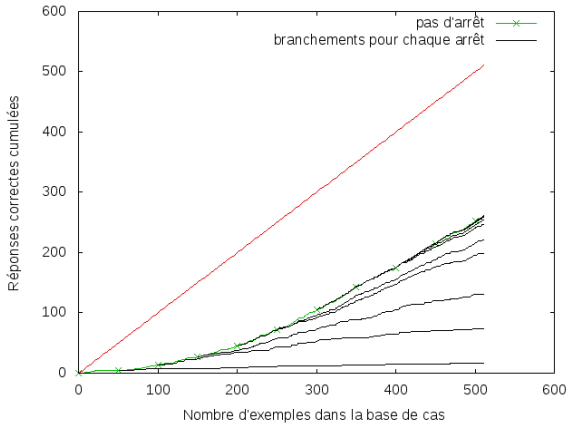


FIGURE 7.3 – Comportement après interruption de l'apprentissage pour une base mélangée

On note immédiatement une grande différence entre la courbe obtenue sur la base groupée par cohortes (figure 7.2) et celle obtenue sur la base mélangée (figure 7.3). En groupant par cohortes, presque aucune réponse correcte n'est donnée par le système après l'arrêt de l'incrément, tandis qu'avec la base mélangée, le système continue à donner des bonnes réponses à un rythme moins élevé dépendant de la taille de la base. Cela peut s'interpréter facilement par le fait que les connaissances sont principa-

lement réutilisées localement lorsque la base est groupée, alors qu'elles tendent à être distribuées de manière équiprobable sur l'ensemble de la base dans le cas du mélange.

Ce résultat montre que le score final obtenu sur la base groupée est fortement dépendant de l'apprentissage incrémental. En effet, un apprentissage hors ligne sur des préfixes de la même séquence d'exemples est incapable de généraliser pour donner des performances comparables. D'autre part, cette expérience montre que la condition dans laquelle la base d'exemples est mélangée rend le système beaucoup plus robuste aux interruptions d'apprentissage. En effet, la variété des exemples permet d'acquérir une densité de connaissances plus importante au cours des premières itérations, par opposition à une densité peu variable et plutôt faible dans le cas des exemples groupés. En outre, cette stabilité est bienvenue puisque c'est la situation de l'ordre pseudo-aléatoire des requêtes qu'on peut s'attendre à rencontrer dans un cas réel d'interaction avec des utilisateurs. Enfin, étant donné que le système est multi-utilisateurs, ces résultats suggèrent qu'il est plus intéressant de recevoir peu d'exemples par utilisateur d'un grand nombre d'utilisateurs plutôt qu'un grand nombre d'exemples de seulement quelques utilisateurs. Ce qui soulève la question : comment l'apprentissage incrémental influence-t-il le comportement du système dans le cas d'un utilisateur nouveau ?

7.5 Introduction d'un utilisateur nouveau

Les requêtes soumises au système sont formulées en langue naturelle, connue pour le grand nombre de variations de surface qu'elle permet pour un même contenu sémantique. On peut donc s'attendre à ce que différents utilisateurs du système aient des styles différents pour la rédaction de leurs instructions pour le système. D'autre part, ils ont certainement des besoins différents en termes de paramètres des commandes. Nous avons évalué la capacité de généralisation du système avec et sans incrément, en soumettant au système les requêtes de la base NOUVEAU après avoir ajouté les exemples de la base QUOTIDIEN à ses connaissances. Les résultats de cette expérience sont décrits dans la figure 7.4. Les performances obtenues précédemment sur la base mélangée (avec incrément), tronquées à 77 requêtes, ont été ajoutés pour comparaison. Les scores finaux obtenus confirment l'utilité de l'apprentissage incrémental : il permet de dépasser l'apprentissage hors ligne de 60%. Le nombre total de réponses correctes est de 16 sur 77 avec incrément, 10 sur 77 sans. Le test précédent sur la base QUOTIDIEN donne seulement 8 réponses correctes sur les 77 premières requêtes, en partant d'une base d'exemples vide. Les coefficients directeurs approchés des courbes avec et sans incrément sont respectivement de $0,35$ et $0,24$ à la fin du test entre les abscisses 60 et 77. Le coefficient pour le test initial sur la base QUOTIDIEN vaut quant à lui $0,18$ sur les mêmes abscisses. On s'attend à ce que le comportement du système avec les requêtes du nouvel utilisateur soit similaire à ce qu'il était avec les requêtes de la base QUOTIDIEN. En effet, bien que différant par leurs styles de rédaction des requêtes, on supposera *a priori* que les performances induites par l'interaction avec différents utilisateurs sont comparables. Ceci suppose bien entendu que leurs besoins en termes de types de requêtes sont similaires, hypothèse que nous posons ici, mais qui dépend des communautés d'utilisateurs.

Les résultats présentés dans la figure 7.4 indiquent qu'un nombre minimal d'exemples pour chaque cohorte est nécessaire avant d'observer une augmentation sensible du score du système. La courbe sur la base non ordonnée de la figure 7.1 semble atteindre sa dérivée finale (ou du moins un coefficient stable jusqu'aux derniers exemples ajoutés) entre la requête 200 et la requête 250, ce qui correspond à avoir vu en moyenne entre 10 et $12,5$ exemples par cohorte (20 au total). Sous cette condition, il n'est effectivement pas possible d'atteindre un score similaire à partir d'une base vide pour le même nombre de cohortes mais sur seulement les 77 requêtes de la base NOUVEAU.

En vue d'exploiter autant que possible chaque nouvel exemple ajouté, il serait intéressant de considérer tous les nouveaux exemples que son ajout permet de générer. La section suivante est ainsi dédiée à l'application de l'extension de la base par analogie, telle que décrite en section 5.5, en contexte

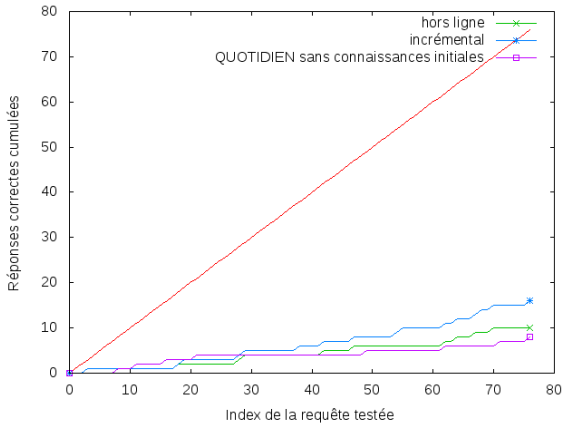


FIGURE 7.4 – Performances avec et sans incrément pour un test sur la base renouvelée, avec la base classique pour connaissances initiales.
La courbe obtenue précédemment pour le test sur la base `bash1` sans connaissances initiales a été ajoutée, tronquée aux 77 premières requêtes, pour comparaison.

incrémental.

7.6 Extension automatique de la base d'exemples

La génération par analogie de nouveaux exemples à partir de la base devrait permettre d'améliorer la performance du système, nous analysons dans cette section son influence relative par rapport à l'apprentissage incrémental. Appliquer la méthode de génération de la section 5.5 dans le contexte de l'apprentissage incrémental revient à effectuer une opération de génération quadratique pour chaque nouvel exemple. Cette opération consiste à former tous les triplets non redondants comprenant le nouvel exemple et deux exemples de la base existante (bases QUOTIDIEN et NOUVEAU confondues, le cas échéant), ce qui correspond à $3 \times |B|^2$ équations analogiques à résoudre. Quand il existe des solutions, elles sont alors ajoutées à la base. D'après le nombre total d'exemples générés donné plus haut dans le tableau 7.1, le nombre moyen d'ajouts à la base pour chaque nouvel exemple est de 126, bien qu'il soit généralement croissant et nul pour les deux premiers exemples. On notera qu'une seule étape de génération est appliquée ; les exemples générés n'ont pas été réutilisés dans les générations ultérieures. L'explosion combinatoire du nombre d'exemples générés que cela impliquerait rendrait rapidement les énumérations trop coûteuses.

La figure 7.5 présente les performances comparées sur la base NOUVEAU avec et sans génération et avec ou sans incrément. La base de connaissances contient initialement l'ensemble des exemples de la base QUOTIDIEN, et les exemples générés associés le cas échéant. L'usage de l'incrément seul ou de la génération seule produisent des résultats similaires. Cependant, plusieurs facteurs influencent la performance obtenue en utilisant la génération, tels que la taille de la base QUOTIDIEN ou la proximité syntaxique entre les requêtes des deux bases, ou plus précisément le nombre de proportions analogiques que l'on peut former entre des requêtes des deux bases. Ces facteurs sont relativement indépendants de l'utilisation ou non de l'incrément, il est donc possible que la proximité des deux résultats ne soit que fortuite. On peut remarquer d'autre part que la génération sans incrément donne de meilleurs résultats que l'apprentissage hors ligne seul, ce qui montre que les connaissances nécessaires pour la production de réponses correctes pour une part des requêtes de la base NOUVEAU étaient déjà contenues implicitement dans la base QUOTIDIEN. La génération par analogie a permis de rendre ces connaissances explicites.

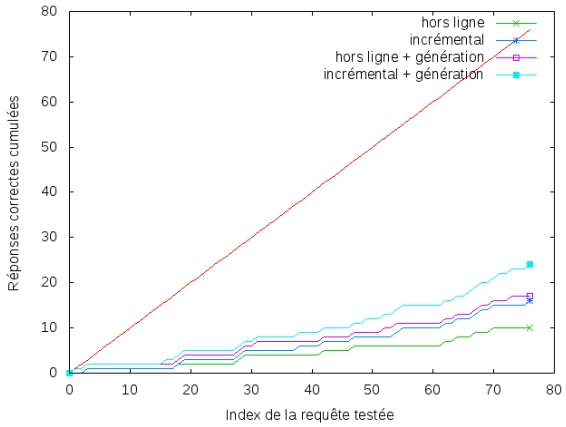


FIGURE 7.5 – Performances comparées avec et sans utilisation de l'incrément et de la génération sur la base renouvelée, avec la base classique pour connaissances initiales.

	avec génération	sans
groupée par cohortes	61,8%	54,2%
groupée par cohortes (ordre inverse)	61,4%	53,9%
mélangée	60,3%	50,9%

TABLEAU 7.2 – Performances finales avec incrément sur la base QUOTIDIEN à partir d'une base vide

La combinaison de la génération et de l'incrément donne de meilleurs résultats que chacun pris séparément, cela indique que ces approches sont au moins en partie complémentaires. D'une part, il n'est pas vraiment surprenant que la génération ne permette pas de produire toutes les réponses correctes données grâce à l'apprentissage incrémental, mais d'autre part, cela signifie que même en ajoutant un à un tous les exemples rencontrés, certaines solutions ne restent accessibles qu'après une étape de génération analogique.

Malgré les 31% du score final combiné, on peut souligner qu'il représente déjà plus de deux fois le score de base de l'apprentissage hors ligne sans génération. Ce ratio prometteur doit cependant être considéré avec prudence à cause de la petite taille de l'ensemble de test.

Le tableau 7.2 reporte les résultats finaux comparés, sans (*idem* figure 7.1) et avec génération de variation. Tous sont obtenus en utilisant l'incrément et avec une base d'exemples initiale vide. On y retrouve également la complémentarité entre génération par analogie et incrément.

7.7 Comparaison avec les bases `R` et `bash2`

Les expériences que nous avons présentées précédemment ont été initialement menées à partir de la seule base d'exemples `bash1`. Nous avons par la suite appliqué le protocole incrémental simple et celui de l'arrêt subit de l'apprentissage aux autres bases collectées décrites au chapitre 2 : `R` et `bash2`. Le protocole d'ajout d'un utilisateur nouveau n'est pas applicable à ces bases. La base `R` ne comporte pas de sous-ensemble collecté de manière indépendante. La base `bash2` est, quant à elle, constituée de l'aggrégation de contributions de 7 utilisateurs. Cependant, chacune de ces contributions prise séparément ne comprend pas suffisamment d'exemples pour produire une expérience comparable à celle menée sur `bash1` pour l'ajout d'un utilisateur nouveau.

Le protocole d'extension des bases par génération analogique n'a à ce jour pas été achevé à cause du coût algorithmique trop élevé de cette opération. Cet état de fait indique que l'utilisation de la

génération en contexte incrémental reste irréalisable en temps réel pour une utilisation du transfert par analogie exhaustif (*i.e.* sans heuristique de recherche) et utilisant une segmentation des énoncés au niveau des mots.

Les figures 7.6 et 7.7 montrent la progression des réponses données par le système lorsque sont soumises incrémentalement toutes les requêtes de la base `R` et `bash2`, respectivement. Ces figures sont analogues à la figure 7.1 et présentent les courbes d'apprentissage dans le cas où les commandes sont groupées par cohortes (approximation), groupées en ordre inverse, et mélangées.

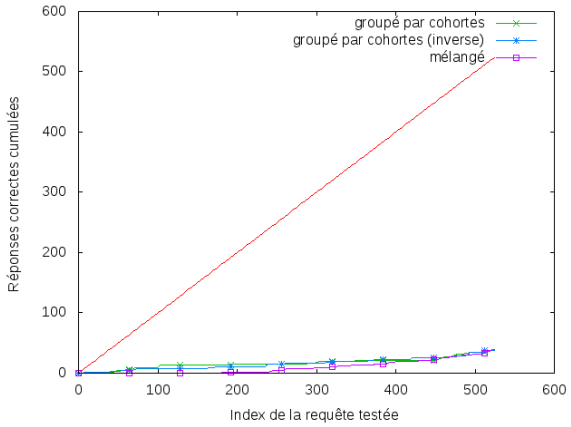


FIGURE 7.6 – Effet de l'ordre des exemples sur les performances de l'apprentissage incrémental sur la base `R`

La première remarque à propos de ces nouvelles figures est la différence notable en terme de performances globales. Les courbes d'apprentissage obtenues sur les bases `R` et `bash2` atteignent des scores bien moins élevés que celles de la figure 7.1 avec respectivement 7 et 16%. Cette différence était attendue concernant la base `R`, car elle est constituée essentiellement d'exemples concernant des commandes différentes par leur nature ou les valeurs de leurs paramètres. Le nombre de régularités reprises avec succès dans des analogies en est donc diminué.

Plusieurs facteurs peuvent expliquer la différence de performance obtenue entre la base `bash2` et la base `bash1` : le nombre accru de commandes distinctes et de formulations différentes des requêtes, toutes deux favorisées par la minimisation des biais lors de la collecte de `bash2`. En effet, `bash2` contient un peu moins de 100 cohortes, contre seulement 20 pour `bash1`. Le nombre de participants différents induit également plus de difficulté à identifier des proportions analogiques dans la base croissante (7 dans `bash2` contre 3 dans `bash1`). Enfin, la formulation implicitement analogique des reformulations de la base `bash1` semble avoir un rôle important dans l'augmentation des résultats.

L'échelle du graphique rend les courbes plus difficiles à lire que sur la figure 7.1, mais on constate la même tendance qu'alors. Le score final ne change pas quel que soit l'ordonnancement des exemples appris incrémentalement. On peut également distinguer une tendance légèrement plus régulière dans la courbe correspondant aux exemples mélangés, mais la granularité des incréments ne permet pas de l'affirmer.

Les figures 7.8, 7.9, 7.10 et 7.11 décrivent respectivement les courbes d'apprentissage obtenues à partir de `R` et de `bash2` avec interruption à tous les 50 exemples suivant le même schéma que présenté en section 7.4. On y retrouve les mêmes tendances qu'identifiées avec la base `bash1` à un changement d'échelle près : le nombre de réponses correctes n'augmente quasiment plus après un arrêt lorsque la

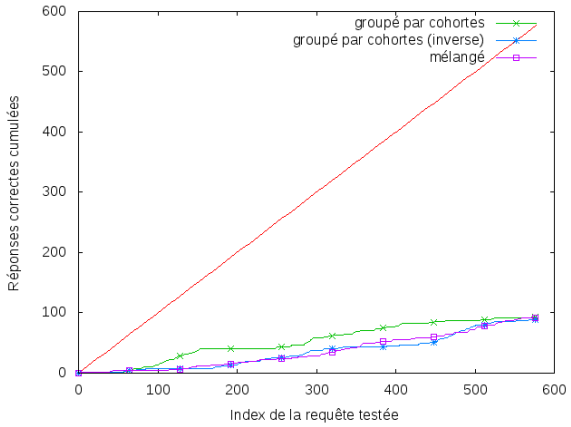


FIGURE 7.7 – Effet de l'ordre des exemples sur les performances de l'apprentissage incrémental sur la base `bash2`

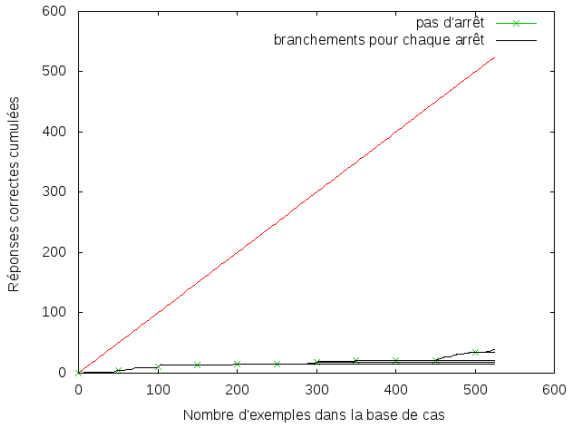


FIGURE 7.8 – Comportement après interruption de l'apprentissage pour une base `R` groupée par cohortes

base est groupée par cohortes, mais il continue de s'améliorer à un rythme moins élevé lorsque la base est mélangée.

On peut noter néanmoins une différence, visible en particulier sur la figure 7.10. Certaines courbes après arrêt de l'incrément continuent à suivre plus longuement la courbe de référence que pour la base `bash1`. Nous attribuons ce comportement à la distribution particulière des cohortes dans la base `bash2` : l'une des cohortes (la commande `cd`) y est sur-représentée à 102 occurrences, alors que le troisième quartile des occurrences est seulement de $4,75$. La commande `cd` étant très simple, peu d'exemples sont nécessaires au système pour produire des réponses cohérentes, ce qui explique que les branchements des courbes aux 50 et 100 premiers exemples suivent la courbe de référence, puis la droite correspondant aux 150 premiers exemples.

7.8 Conclusion

L'utilisation du raisonnement analogique dans le contexte de l'apprentissage incrémental permet, pour la base d'exemples `bash1` et sans connaissance initiale, de dépasser les 50% de bonnes réponses avec seulement 512 exemples. La pente moyenne sur les 200 dernières requêtes pour la base mélangée

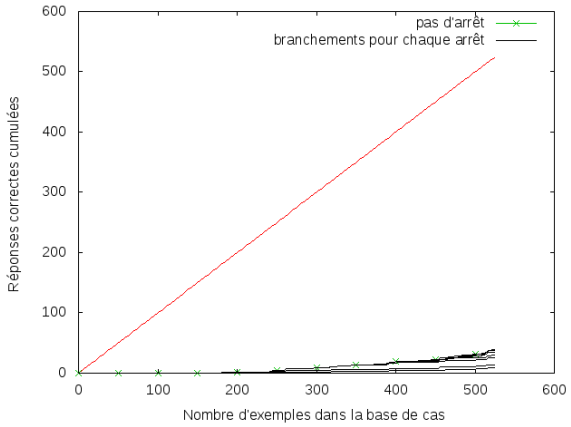


FIGURE 7.9 – Comportement après interruption de l'apprentissage pour une base `R` mélangée

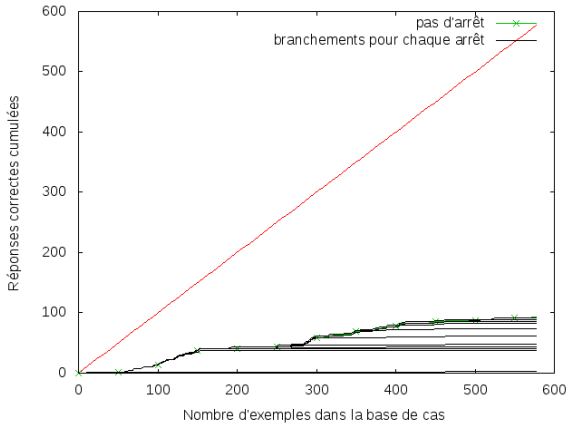


FIGURE 7.10 – Comportement après interruption de l'apprentissage pour une base `bash2` groupée par cohortes

atteint $0,74$, ce qui correspond donc à un taux de bonnes réponses de 74% sur ce segment. Cependant, ce résultat positif est très sensible aux conditions d'utilisation, en particulier à la distribution des commandes (cohortes) et à la diversité des formulations en langue naturelle.

En effet, nous avons vu qu'une moyenne de 10 à $12,5$ exemples par cohorte est nécessaire avant que le système n'atteigne un taux de réponses correctes stable pour cette cohorte, ce taux se situant donc à un peu moins de 74% pour le cas de la base `bash1`. Or, cette base d'exemples présente une distribution des cohortes équilibrée du fait du protocole de collecte utilisé, ce qui n'est pas le cas des bases `R` ni `bash2`. Par ailleurs, ces deux bases contiennent une variation linguistique importante, due à des reformulations systématiques pour la première, et à une collecte auprès de sept utilisateurs différents pour la seconde.

Des tests à plus grande échelle permettraient, avec les résultats sur `bash1`, d'identifier la relation entre le nombre de cohortes, la forme de leur distribution, et le coefficient de la courbe d'apprentissage.

Concernant la comparaison des différents ordonnancements de la base, alors que les tests sur la base groupée voient leur taux moyen (dérivée des courbes présentées) diminuer ou stagner, le taux de bonnes réponses dans le cas réel d'une base non ordonnée est en constante augmentation. Malgré cela, la performance finale est meilleure pour la base groupée. La raison de cette différence n'est pas

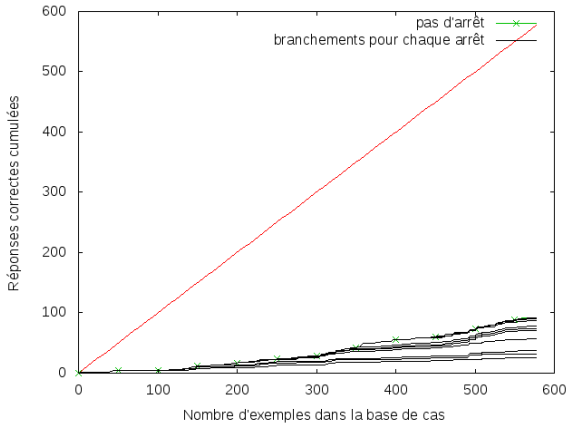


FIGURE 7.11 – Comportement après interruption de l'apprentissage pour une base `bash2` mélangée

triviale. En effet, d'une part, la symétrie de l'analogie permet, comme discuté plus haut, de toujours résoudre au moins une requête parmi les quatre d'une proportion valide dans la base. Mais d'autre part, les proportions entre exemples de la base sont nombreuses et comptent des intersections, des exemples peuvent ainsi jouer dans plusieurs résolutions analogiques. On peut définir un ordre sur les exemples de la base grâce au nombre de proportions dans lesquelles chacun joue un rôle. Un exemple jouant dans plus de proportions qu'un autre sera dit plus informatif. Ainsi, l'ordonnancement optimal d'une base donnée consiste à placer les exemples les plus informatifs en premier. Il est donc surprenant que tous les ordonnancements mélangés que nous avons essayés pour `bash1` aient des score moins élevés qu'avec la base ordonnée. Cependant, l'absence de différence pour les autres bases (voire dans certains cas une légère avance du cas mélangé) laisse supposer que l'ordonnancement groupé de la base `bash1` n'est proche de l'ordonnancement optimal que de manière fortuite.

Bilan et pistes à l'étude

Sommaire

Nous présentons dans ce chapitre un bilan de la deuxième partie, en reprenant les approches proposées dans les chapitres 5 à 7 et fondées sur le modèle d'analogie formelle présenté dans le chapitre 4. Nous rappelons brièvement les principes et motivations de ces approches pour ensuite revenir sur leurs enjeux. Enfin, dans la section 8.2, nous discutons d'une piste en cours d'exploration pour la réduction du coût d'exécution des algorithmes, et qui est fondée sur la segmentation informée des séquences.

8.1 Bilan

La parcimonie des exemples disponibles nous a conduit au choix d'une méthode de transfert de domaine à base d'exemples : le raisonnement par analogie formelle. En tant que méthode de raisonnement à base d'exemples, il s'agit d'une approche intelligible, permettant donc de retracer précisément et de manière aisément compréhensible les opérations conduisant à tout résultat. L'analogie est en outre souvent présentée comme un élément important de notre cognition symbolique. Déjà utilisée en traduction automatique, l'analogie présente l'avantage de produire des résultats cohérents dès un très petit nombre d'exemples, comme l'ont confirmé nos résultats. De plus, malgré une couverture variable imputable à la couverture nécessairement faible de l'ensemble des cas possibles par nos bases d'exemples restreintes, les expériences ont montré un taux de bonnes réponses particulièrement élevé parmi les réponses produites par le système.

8.1.1 Transferts direct et indirect

La spécificité du transfert de langages depuis une langue naturelle vers un langage formel, par opposition au transfert (traduction) entre deux langues naturelles est qu'il est très fréquent qu'un ou plusieurs éléments de l'énoncé en langue naturelle se retrouvent à l'identique dans l'expression en langage formel. Il s'agit des paramètres de la requête. Nous avons montré que cette spécificité rend pertinente l'utilisation conjointe d'une approche de transfert direct et d'une autre de transfert indirect. Ces approches par analogie sont complémentaires en ce que leurs avantages et faiblesses sont duales l'une de l'autre. Nos expériences ont montré que la variation des paramètres des requêtes, qui fait drastiquement chuter le score du transfert indirect, est traitée dans certains cas par le transfert direct. Inversement, lorsque les paramètres des requêtes ne varient pas (*i.e.* ils sont déjà présents dans des exemples de la base), le score du transfert indirect est bien meilleur, tandis que le transfert direct est inutile.

8.1.2 Extension des bases

Dans le but d'étendre la couverture des bases d'exemples, nous avons envisagé d'augmenter par raisonnement analogique le nombre d'exemples disponibles. Le choix de l'analogie pour cette opération a été motivé par l'observation de nombreux schémas analogiques parmi les exemples rédigés par les participants lors des collectes d'exemples. Cette méthode de génération par fermeture transitive analogique a permis de produire des bases d'exemples de 40 à 97 fois plus grandes que les bases initiales. Les performances du système ont été améliorées grâce à ces nouveaux exemples, mais dans une proportion relativement faible au regard de l'accroissement de la base elle même. Sachant que le score augmente d'un peu moins de 10 points dans l'ensemble, le ratio nouvelles bonnes réponses sur nouveaux exemples générés est compris entre $1 \cdot 10^{-3}$ et $2,5 \cdot 10^{-3}$. La génération d'exemples par analogie ne produit donc pas beaucoup d'exemples informatifs en proportion, à tout le moins lorsque l'on considère la fermeture transitive d'ordre 1 seulement, comme nous l'avons fait.

La génération par analogie a pour effet de précalculer la totalité des solutions des analogies indirectes identifiées dans la base initiale. Celles-ci deviennent donc triviales à identifier, et d'autres solutions directes et indirectes deviennent accessibles.

Nous n'avons pas tenté une génération d'exemples d'ordre 2 à partir des bases générées car le transfert de langage par raisonnement analogique formel est une méthode coûteuse. La taille des bases obtenues à l'ordre 1 induit déjà une durée d'exécution élevée de l'algorithme, de l'ordre de plusieurs dizaines de secondes en moyenne. Ces délais ne sont pas satisfaisants pour une utilisation réellement interactive et on ne peut donc envisager l'utilisation de bases plus importantes en l'état. En effet, si l'on s'en tient au taux d'accroissement obtenu par fermeture transitive d'ordre 1, on peut s'attendre à construire une base d'ordre 2 de 400 à $9\ 400$ fois plus grande que la base initiale (à l'ordre 0).

Par ailleurs, une proportion importante [1] des requêtes générées ne font pas sens pour un lecteur humain, et sont donc très peu susceptibles d'intervenir dans une analogie productive lors du transfert d'une requête de l'utilisateur. Notons que cette remarque s'appuie sur la conjecture que seules les requêtes faisant sens pour l'humain sont susceptibles d'intervenir dans des proportions analogiques productives. Rien ne permet *a priori* d'affirmer que le contraire soit impossible, cependant cette conjecture nous semble raisonnable au vu de l'amélioration modérée qu'à permis l'utilisation de la base générée.

8.1.3 Relaxation

Le raisonnement par analogie formelle appliqué aux séquences de caractères ou de tokens est une méthode stricte qui ne permet pas d'identifier des réponses approchées. Cette propriété permet la précision élevée qu'on a pu observer, mais a également un impact négatif sur le silence du système. Afin de réduire ce silence, nous avons envisagé de relâcher les contraintes fortes sur l'opération de résolution d'équations analogiques ainsi que sur la recherche d'analogies dans la base.

Pour relâcher les contraintes sur l'opération de résolution, nous avons proposé de réutiliser une version simplifiée d'un algorithme approché proposé par Laurent Miclet. Pour la relaxation de la recherche d'analogies dans la base, nous avons modifié l'algorithme optimisé de recherche indexée à l'aide des arbres de comptage. Chacune de ces relaxations a permis de réduire le silence du système, mais le coût en termes de précision n'est pas le même. La résolution d'analogies concerne directement la production des solutions qui seront proposées par le système, la relaxation à ce niveau produit beaucoup de commandes invalides en permettant le transfert "forcé" depuis la requête vers la commande d'éléments propres à la langue naturelle. En revanche, la relaxation de l'opération de recherche d'analogies, applicable uniquement lors du transfert indirect, permet des approximations au sein des requêtes en langue naturelle uniquement. Parmi ces approximations, certaines sont légitimes du point de vue lexical, syntaxique ou sémantique, d'autres ne le sont pas. Cependant, les approximations en langue naturelle n'ont pas d'influence négative sur la production de commandes tant que la déviation reste exclusive au domaine source. Elles ne font donc que permettre la production d'un plus grand nombre de réponses.

Cette efficacité a cependant une contrepartie : nous avons noté que ces relaxations de contraintes ne viennent pas sans une augmentation non négligeable de la combinatoire des cas à considérer, et donc du temps d'exécution. L'augmentation est en réalité exponentielle en fonction de la déviation Δ, mais bornée par le coût de l'algorithme sous-jacent appliqué exhaustivement, c'est-à-dire l'exploration complète de l'arbre de comptage pour une déviation sur la langue naturelle, et Δ exécutions de la résolution analogique pour une déviation sur le langage formel.

8.1.4 Incrément

Comme annoncé dans la problématique de notre travail, nous avons étudié le comportement du système apprenant en contexte incrémental. En considérant que l'utilisateur consciencieux enseigne au système la solution correcte à chaque fois que le système produit une erreur ou reste silencieux et

1. Cette proportion n'a pas été mesurée de manière exhaustive sur les bases générées, car l'annotation est très coûteuse.

supprime le fichier dans `dir/`	:	imprime le document dans `print_dir/`
	::	
supprime les fichiers dans `dir/`	:	?

EXEMPLE 8.1 – Équation analogique sans solution pour une segmentation au niveau des tokens, mais ayant une solution pour une segmentation au niveau des caractères.

en enrichissant ainsi une base d'exemples initialement vide, l'ajout progressif des exemples à la base permet au système d'atteindre jusqu'à 60% de réponses correctes cumulées pour des requêtes toujours nouvelles. Comme pour les autres expériences, le nombre de réponses incorrectes reste très faible, voire nul dans la plupart des tests.

Nous avons pu constater que le cas dans lequel les exemples sont ajoutés dans un ordre aléatoire, supposé plus naturel qu'un ordre trié, permet une bien meilleure stabilité de la performance du système lorsqu'il est sujet à des interruptions subites de l'incrément, c'est-à-dire lorsqu'il interagit avec un utilisateur qui, en cas d'échec, ne peut pas lui enseigner les commandes attendues. Par ailleurs, l'ordonnancement aléatoire donne également le meilleur coefficient d'apprentissage à terme. L'évolution du coefficient d'apprentissage au cours de l'incrément a montré que la performance optimale semble être atteinte après 10 à 12 exemples donnés pour chaque commande. Ce nombre est en outre susceptible de diminuer pour les nouvelles commandes, du fait de la redondance croissante des formulations en langue naturelle dans la base d'exemples.

8.2 Pistes de segmentation à l'étude

Tout au long des travaux présentés dans ce document, nous avons appliqué le raisonnement analogique sur des séquences de tokens. La segmentation au niveau du caractère a été envisagée mais rapidement écartée à cause de son surcoût algorithmique important, pour un apport en solutions pertinentes très faible (étant donné nos jeux de test). Nous ouvrons dans cette section sur des alternatives en matière de segmentation des séquences pour leur traitement efficace par raisonnement analogique formel.

8.2.1 Motivation et enjeux

L'objectif de concevoir une méthode de segmentation meilleure qu'au niveau des tokens est double. D'une part, on sait qu'un nombre plus faible de segments à considérer conduit à une réduction du coût algorithmique des opérations analogiques. Une telle réduction serait la bienvenue au vu des problèmes de temps d'exécution que nous avons rencontrés tout au long de nos travaux. D'autre part, il peut exister des solutions analogiques pour un triplet segmenté au niveau du caractère, qui n'existent pas pour ce même triplet segmenté au niveau des tokens, comme l'illustre l'exemple 8.1. Les bases d'exemples que nous avons collectées et utilisées ne contiennent pas beaucoup de ces cas, mais il est fréquent d'en rencontrer dans l'utilisation de la langue naturelle, suivant les flexions des formes.

Concernant le langage formel, il est difficile de prévoir la fréquence de cas de ce type, car elle dépend des besoins de l'utilisateur, de ses habitudes. En effet, rappelons que, dans nos expériences, l'essentiel des équations analogiques formées exclusivement dans l'espace des commandes relèvent des formes canoniques $[X : X :: Y : Y]$ ou $[X : Y :: X : Y]$. Envisager une segmentation à un autre niveau que celui du token pour les commandes n'est donc intéressant principalement que pour rendre compte de schémas réguliers dans la forme des paramètres (voir l'exemple 8.2), ou bien lorsque les paramètres ne sont pas séparés du reste de la commande par des caractères d'espacement en tant que tokens à part entière. Cependant, cette exception peut souvent être évitée par l'insertion automatique

ouvre le rapport du 30/08/16	:	`mimeopen rapport_300816.md`
	::	
ouvre le rapport du 19/12/16	:	`mimeopen rapport_191216.md`

EXEMPLE 8.2 – Proportion analogique mixte (transfert direct) requérant la segmentation au niveau des caractères.

d'espacements aux frontières des opérateurs et des symboles du langage. La syntaxe des opérateurs et des symboles diffère selon les langages, cette opération se fait donc au prix de l'indépendance au langage de sortie. Cependant, on peut considérer cette indépendance au langage de sortie comme moins critique par comparaison à celle au langage d'entrée. En effet, le premier est un langage formel dont la syntaxe est connue de manière complète [2] puisqu'elle a été construite tandis que le second est une langue naturelle, donc ambigüe et partiellement modélisée. Nous envisageons la segmentation dynamique uniquement sur le domaine des requêtes. En effet, nous nous focalisons en priorité sur l'application la plus susceptible de montrer des résultats significatifs, une segmentation dynamique sur le domaine des commandes étant peu à même d'apporter une différence notable du fait, notamment, du nombre plus restreint de tokens dans les commandes.

8.2.2 Approches

Segmentation en ligne exhaustive

Les approches les plus génériques impliquent de décider de la segmentation appliquée à un énoncé uniquement lors de son utilisation dans une proportion ou une équation analogique. Il n'existe donc pas de segmentation *a priori*, c'est pourquoi nous nommons cette stratégie la segmentation "en ligne". La version naïve de cette stratégie implique d'énumérer, pour chaque quadruplet ou triplet de séquences selon que l'on vérifie ou résout l'analogie, l'ensemble des segmentations conjointes possibles de deux des séquences, jusqu'à trouver une solution ou épuiser les segmentations.

Le désavantage immédiat de cette méthode est sa complexité. Une séquence de longueur $|s|$ (en nombre d'atomes, par exemple les caractères) compte $2^{|s|-1}+1$ segmentations possibles. Le nombre de segmentations conjointes possibles pour une paire de séquences (s_1, s_2) est donc de l'ordre de $2^{|s_1|+|s_2|-2}$. C'est par ce facteur effrayant que l'on doit *multiplier* la complexité algorithmique de recherche, déjà quadratique ou cubique en fonction de la taille de la base.

Bien entendu, nous avons implémenté des heuristiques. D'une part, la recherche de segmentations compatibles pour l'analogie est effectuée dans l'ordre de la plus globale à la plus détaillée, c'est-à-dire en testant les segmentations à 1 segment, puis 2, puis 3, etc. En effet, l'objectif premier d'utiliser la segmentation dynamique est de réduire le nombre de segments, autant donc commencer par les segmentations les plus réduites ; ce sont également celles qui sont les moins nombreuses (il y a $\binom{|s|}{k}$ segmentations à k segments). D'autre part, la recherche est abandonnée si aucune segmentation compatible n'est trouvée avant un nombre défini d'itérations I.

Notons que, malgré cette complexité algorithmique extrême, les heuristiques introduites pourraient théoriquement conduire à un gain de cette stratégie par rapport à la recherche simple d'analogies dans le cas très précis dans lequel le coût de résolution/vérification analogique au niveau de segmentation par défaut est supérieur au coût de l'énumération (limitée) des segmentations pour le triplet/quadruplet concerné. Cette possibilité théorique ne s'est pas confirmée en pratique, car la possibilité de gain ne concerne que le cas positif, dans lequel une solution analogique existe. Dans le cas négatif, la perte est certaine puisqu'on effectue toujours I itérations, contre 1 dans le cas de la recherche simple.

2. Par exemple, la grammaire complète du langage `shell` POSIX (un sous-langage de `bash`) peut être trouvée à cette adresse : pubs.opengroup.org/onlinepubs/9699919799/utilities/V3_chap02.html#tag_18_10

Segmentation en ligne partielle

Afin d'éviter un parcours systématique et trop coûteux des segmentations possibles de l'énoncé d'entrée et des énoncés de la base d'exemples, nous nous sommes inspiré des heuristiques déjà utilisées [Somers *et al.*, 2009] pour la recherche de proportions analogiques valides dans la base. L'une de ces heuristiques propose de tenter en priorité de constituer des analogies avec les exemples de la base ayant les plus longues sous-chaînes communes avec l'énoncé à transférer. L'idée est qu'au lieu de définir, comme précédemment, une segmentation complète des énoncés correspondant exactement aux frontières des cofacteurs de la proportion analogique, on envisage de ne regrouper que le segment correspondant à la plus longue sous-chaîne commune identifiée lors de la recherche. Le reste de l'énoncé reste segmenté au niveau de granularité choisi (caractères ou tokens dans le cas présent), voir exemple 8.3.

r :	Retourne dans le				home	.		
e_1 :	Retourne dans le				dossier	parent	.	
e_2 :	Déplace	toi	dans	le	home	.		
e_s :	Déplace	toi	dans	le	dossier	parent	.	

EXEMPLE 8.3 – Segmentation partielle de quatre énoncés et alignement en proportion analogique. La double séparation indique la frontière entre cofacteurs de l'analogie. r est la requête entrée par l'utilisateur, e_1 est un énoncé de la base ayant une plus longue sous-chaîne commune avec r, e_2 est un autre énoncé de la base, et e_s est la solution de l'équation $[r : e_1 :: e_2 :?]$

L'hypothèse motivant l'utilisation de cette méthode est qu'elle permet à la fois de pré-sélectionner au moins l'un des trois énoncés recherchés pour constituer la proportion, et d'en proposer une segmentation plausible pour la constitution de proportions analogiques, ce qui doit diminuer le temps nécessaire pour produire la solution. Elle permet également lorsque c'est pertinent, de découper le segment correspondant à la plus longue sous-chaîne commune à un niveau de granularité arbitraire, c'est-à-dire potentiellement au niveau du caractère. Par ailleurs, l'intérêt de cette méthode réside également dans le coût algorithmique particulièrement bas de la procédure de recherche de la plus longue sous-chaîne commune. En effet, l'algorithme proposé par Ukkonen [Ukkonen, 1995] permet, après construction d'une structure d'indexation de la base appelée arbre de suffixes généralisé, de parcourir cette structure pour identifier tous les énoncés de la base ayant une plus longue sous-chaîne commune avec la requête soumise en un temps linéaire en fonction de la longueur de ladite requête. On remarquera que l'ensemble des énoncés de la base ayant une plus longue sous-chaîne commune avec la requête r contient généralement plus d'un élément pour deux raisons :

- la même plus longue sous-chaîne peut apparaître en de multiples endroits d'un ou de plusieurs énoncés de la base,
- r peut avoir des plus longues sous-chaînes communes distinctes avec différents énoncés de la base, mais de même longueur.

Nous retiendrons tous les énoncés de la base ayant une plus longue sous-chaîne commune avec r de longueur maximale égale à L.

Nous avons donc révisé la procédure de transfert indirect (*cf.* section 5.3) pour y utiliser la recherche de la plus longue sous-chaîne commune [3] lors de la recherche de proportions analogiques dans l'espace

3. Remerciements à Guillaume Dubuisson Duplessis dont nous avons pu utiliser l'implémentation `gstlib` de l'algorithme de Ukkonen.

source, *i.e.* l'espace des requêtes :

1. étant donné la requête r soumise par l'utilisateur, on recherche toutes les occurrences des plus longues sous-chaînes communes de longueur maximale L entre r et les énoncés de la base ;
2. pour chacune des plus longues sous-chaînes communes identifiées, resegmenter r et l'énoncé concerné e_1 pour en tenir compte, comme illustré dans l'exemple 8.3 ;
3. itérer sur l'ensemble des énoncés de la base pour instancier e_2 ;
4. résoudre $[r : e_1 :: e_2 :?]$; si une solution existe et la solution s de degré minimal appartient également à la base, poursuivre la procédure de transfert avec la résolution de l'équation analogique formée avec c_1, c_2 et c_s, les commandes respectives associées à e_1, e_2 et e_s dans la base d'exemples.

Nous n'avons pas appliqué la recherche de la plus longue sous-chaîne commune à la procédure de transfert direct. Le gain attendu dans le cas du transfert direct est moins important qu'avec le transfert indirect. En effet, on ne fait pas appel à la procédure coûteuse de recherche de proportions analogiques, la marge d'amélioration est donc plus réduite.

	procédure initiale (5.7.1)	procédure révisée (`gstlib`)
réponses correctes	36	34
précision	1	1
temps par requête moyen (exhaustif)	$0,81$	$2,53$

TABLEAU 8.1 – Comparaison de la procédure de transfert indirect initialement développée avec la procédure utilisant la recherche de la plus longue sous-chaîne commune.

L'application de la procédure indirecte révisée décrite à la base d'exemples `bash1` a permis d'obtenir des résultats similaires en termes de réponses données, le tableau 8.1 les décrit brièvement. La procédure révisée retourne deux bonnes réponses de moins que la procédure initiale. Malgré l'agglomération d'un segment potentiellement plus grand correspondant à la plus longue sous-chaîne commune, la procédure révisée rétablit pour le reste de la requête et de l'énoncé une segmentation au niveau des tokens. De nouvelles bonnes réponses peuvent être apportées par cette méthode lorsqu'une segmentation au niveau des mots est insuffisante pour établir une proportion analogique, comme illustré dans l'exemple 8.4. Ce cas n'a pas été rencontré avec l'ensemble de test QUOTIDIEN de `bash1`.

r :	Supprime	ϵ			**f1**	**.txt** sans demander de confirmation
e_1 :	Efface	ϵ			**f2**	**.txt** sans demander de confirmation
e_2 :	Supprime	le	fichier	texte	f1	ϵ
e_s :	Efface	le	fichier	texte	f2	ϵ

EXEMPLE 8.4 – Cas d'une segmentation induite par la plus longue sous-chaîne commune permettant la production d'une réponse correcte là où une segmentation au niveau des tokens ne le permet pas. Les deux tokens sous-découpés sont présentés en gras. À nouveau, les cofacteurs de l'alignement analogique sont délimités par les doubles séparateurs.

En revanche, certaines bonnes réponses produites grâce à la procédure initiale peuvent ne pas être retrouvées en utilisant la pré-sélection de la plus longue sous-chaîne commune. Ce cas survient

e_1 :	trouve le tar de	nelda_v1.5	dans le répertoire nelda
e_2 :	trouve dans le répertoire nelda l'archive de	nelda_v1.5	
e_3 :	trouve le tar de	nelda la v1.5	dans le répertoire nelda
r :	trouve dans le répertoire nelda l'archive de	nelda la v1.5	

EXEMPLE 8.5 – Exemple de proportion analogique constituée par la procédure initiale (segmentation au niveau des tokens) et produisant une réponse correcte. Les plus longues sous-chaînes communes trouvées pour la requête r sont de longueur 51, par exemple avec l'énoncé "trouve dans le répertoire nelda l'archive de nelda_v1.5", alors que la plus longue sous-chaîne commune avec l'un des énoncés $\{e_1, e_2, e_3\}$ est de longueur 46.

lorsqu'aucune des proportions analogiques permettant de produire la bonne réponse n'inclut d'énoncé ayant une plus longue sous-chaîne commune avec la requête soumise. Les proportions analogiques mises en jeu dans nos travaux et ailleurs dépassant très rarement[4] le degré 3, les facteurs tendent à avoir une grande taille relativement à la longueur des séquences qu'ils découpent. En conséquence, on s'attend à ce que ce cas de défaut de réponse par la procédure révisée soit en général peu fréquent. Les résultats obtenus en montrent néanmoins deux occurrences sur l'ensemble de test utilisé. Il s'agit en réalité de deux fois le même, que nous avons reporté dans l'exemple 8.5.

Rappelons que la motivation première pour l'utilisation d'une méthode de segmentation informée est la réduction du temps de traitement, grâce à la diminution du nombre de segments à traiter pour chaque vérification ou résolution analogique. Sous cet angle, les résultats montrés dans le tableau 8.1 sont décevants, car ils mettent en évidence au contraire un accroissement du temps par requête moyen. Cependant, notre implémentation de la procédure de recherche révisée identifie les plus longues sous-chaînes communes à partir des énoncés découpés au niveau du caractère et non du token. Ce n'est qu'ensuite qu'une segmentation est déterminée comme dans les exemples 8.3 et 8.4. Ce choix d'implémentation a été guidé par l'objectif de permettre la résolution de cas supplémentaires comme nous l'avons montré avec l'exemple 8.4. Le découpage initial en caractères entraîne un surcoût et rend difficile l'alignement des deux mesures en termes de durée. À titre de comparaison, la procédure initiale exécutée avec une segmentation au niveau des caractères donne une durée moyenne par requête de $6,02$ secondes, tout en produisant les mêmes réponses qu'avec la segmentation au niveau des mots. Là encore, l'alignement reste difficile à cause des différences de traitement entre la procédure initiale et la procédure révisée. En particulier, la procédure révisée fait appel à la résolution d'une équation analogique, alors que la procédure initiale n'opère ni résolution ni vérification, mais fait en revanche une recherche dans l'arbre de comptage (*cf.* section 5.3).

A posteriori, l'absence manifeste de cas comparables à l'exemple 8.4, nécessitant un découpage initial au niveau des caractères pour produire une réponse correcte, suggère que l'implémentation de la procédure révisée avec une segmentation directement basée au niveau des tokens devrait également être intéressante. Il s'agit d'une piste encore à explorer.

Segmentation hors ligne

La seconde stratégie pour la segmentation dynamique consiste à déterminer *a priori* une ou plusieurs segmentations pour les énoncés, sans tenir compte des analogies qui peuvent être formées au moment du transfert. Il s'agit donc d'une stratégie "hors ligne". La segmentation au niveau des tokens telle que nous l'avons utilisée dans l'essentiel de nos travaux relève de cette stratégie, de même que la

4. Nous n'avons malheureusement pas de statistiques précises pour étayer cette affirmation, qui n'est donc fondée que sur notre observation répétée sur de nombreux échantillons. La distribution des degrés des analogies dans nos expériences sera calculée et intégrée au document final.

segmentation au niveau des caractères. Bien entendu, ce qui nous intéresse ici est de concevoir une approche plus fine que chacune des deux précédentes afin de limiter le nombre total de segments tout en conservant des frontières entre segments cohérentes au regard des proportions analogiques à constituer.

Il semble que nous ayons épuisé les méthodes indépendantes de la langue pour envisager la segmentation hors ligne, l'option restante consiste donc à utiliser des connaissances sur la langue utilisée afin de segmenter les énoncés. On peut pour cela faire appel à un analyseur syntaxique existant. On choisit alors un niveau de profondeur dans l'arbre d'analyse retourné pour délimiter les segments de chaque énoncé.

Outre le fait que l'utilisation de cette approche contraint à abandonner l'indépendance à la langue, on peut s'attendre à ce que la qualité de l'analyse syntaxique ne soit pas optimale. La raison principale est que les analyseurs syntaxiques ne sont pas nécessairement adaptés au traitement d'énoncés sous cette forme, en particulier lorsqu'ils contiennent les paramètres de la requête qui peuvent être sous une forme inconnue, ou bien au contraire sous une forme lexicale (*e.g. "imprime le fichier **zxt35.pdf** et le fichier texte appelé **article**"*).

Nous avons utilisé l'étiqueteur morpho-syntaxique TreeTagger en tant que segmenteur (ou *chunker*) simple sur les énoncés des requêtes des bases d'exemples. TreeTagger associe des étiquettes morpho-syntaxiques, parfois hiérarchiques à des sous-parties des énoncés, qui définissent alors des frontières que nous avons utilisé avec deux types de découpage. Pour le premier type de découpage, nous avons extrait tous les nœuds de profondeur $n-1$ et fusionné les nœuds fils s'ils existaient, n étant la profondeur de l'arbre d'analyse considéré. Le second type de découpage constitue les segments à l'aide des nœuds de profondeur n, les suites de tokens contigües n'étant pas annotées au niveau n sont également considérées comme des segments indépendants.

8.2.3 Bilan et pistes

La segmentation en ligne est la stratégie permettant de préserver au mieux les résultats produits par le système tout en permettant une subdivision pertinente des tokens lorsque cela est utile. La segmentation hors ligne, telle que nous l'avons envisagée jusqu'à présent, ne permet que des découpages au niveau du token ou plus large, et la rigidité des segmentations produites à l'aide d'analyseurs syntaxiques non spécialisés cause la perte de résultats corrects, par ailleurs produits avec une segmentation au niveau des tokens. En revanche, bien que les expériences menées ne montrent pas d'amélioration de la durée d'exécution par rapport à une segmentation sur les tokens, la stratégie hors ligne ne souffre pas de l'explosion combinatoire propre à la stratégie en ligne exhaustive. La stratégie exhaustive s'est montrée clairement irréalisable en l'état, mais on peut espérer qu'un élagage drastique de l'exploration rende les délais d'exécution plus raisonnables. Un tel élagage peut être envisagé car nous avons observé que l'essentiel des analogies formées comptent trois facteurs ou moins.

Les résultats obtenus jusqu'à présent à l'aide de la segmentation en ligne partielle par plus longue sous-chaîne commune ne correspondent pas aux attentes. Nous ne pouvons pas en tirer de conclusion claire avant d'avoir établi une explication pour ces résultats contradictoires.

Il reste à l'évidence de nombreuses pistes ouvertes pour l'exploration des stratégies de segmentation appliquées au raisonnement analogique sur les séquences, et il nous semble important de poursuivre dans cette direction au vu de l'allègement de la charge de calcul analogique que permet une segmentation appropriée.

Troisième partie

Conclusion

Conclusion générale et perspectives

Sommaire

Le travail présenté dans ce document vise à l'intégration du champ des systèmes assistants et de celui des systèmes apprenants pour la conception d'un assistant opérationnel incrémental. Cette intégration permet d'exploiter les modalités offertes par l'un des domaines en vue d'améliorer la performance concernant l'autre, et réciproquement. En particulier, l'apprentissage permet de doter le système assistant d'une capacité de progrès vis-à-vis de ses utilisateurs, et l'interaction permet au système apprenant d'obtenir les exemples les plus pertinents selon les besoins réels des utilisateurs.

Nous avons défini un mode d'interaction adapté à ce type de système et utilisé ce protocole comme point de départ pour envisager les méthodes d'apprentissage. En particulier, nous nous sommes intéressé aux problèmes posés pour la conception d'un assistant acceptant des requêtes de l'utilisateur en langue naturelle, et effectuant des actions adaptées. Nous avons donc modélisé la tâche sous forme d'un problème de transfert de langages, à partir d'une langue naturelle vers un langage formel.

Les contributions apportées au long de nos travaux sont synthétisées dans la section ci-dessous, puis nous ouvrons sur les perspectives qui en découlent, directement ou indirectement.

9.1 Contributions

9.1.1 Recueil de bases d'exemples

Les données d'apprentissage pour une tâche de transfert de langage sont constituées d'associations entre les requêtes d'utilisateurs en langue naturelle, et les commandes associées dans un langage de programmation. Extraire, à partir des données existantes à travers le Web, une base d'exemples fiable est difficile, principalement à cause de leur hétérogénéité. En effet, les ressources existantes pouvant s'apparenter à des associations entre requêtes et commandes, tels que les cours, forums d'aide, tutoriels et documentations sont très divers et insuffisamment normés. La raison en est qu'ils sont avant tout destinés aux humains dont l'arsenal cognitif pour les interpréter dépasse de loin la portée d'un système assistant relevant de l'intelligence artificielle dite faible. Il s'agit donc d'un problème de recherche à part entière, que notre sujet ne couvre pas.

Nous avons collecté plusieurs bases d'associations entre requêtes et commandes auprès de volontaires avec pour objectif de maximiser la représentativité de ces bases par rapport à l'utilisation hypothétique du système. Nous avons ainsi constitué une base associant des requêtes en français à des commandes en `R` à partir d'un site didactique et une base associant des requêtes en français à des commandes en `bash` grâce aux reformulations de participants informés. Afin d'écarter les biais liés à l'information du participant et au contexte de collecte, nous avons mis en place un protocole nouveau fondé sur l'extraction directe de commandes réellement utilisées depuis les sessions informatiques des participants. Ce protocole nous a finalement permis de constituer une dernière base, également du français vers le `bash`, plus diverse et représentative que les deux précédentes.

9.1.2 Transfert depuis la langue naturelle vers le langage formel

Compte tenu du nombre relativement faible de commandes possibles, sans tenir compte de leurs paramètres, relativement au nombre de requêtes en langue naturelle, nous avons en premier lieu proposé une approche de transfert fondée sur la similarité lexicale des requêtes. Pour cette approche, nous avons admis l'hypothèse que la similarité est transitive entre le domaine source et le domaine cible du transfert. Autrement dit, deux requêtes très proches sont également hautement susceptibles de correspondre à la même commande. Cette hypothèse est inexacte (*cf.* section 3.2.1), mais les contre-exemples sont exceptionnels.

L'approche par similarité a permis d'obtenir une précision à 3 supérieure à 50%, et nous avons pu atteindre 60% grâce à une combinaison des différentes similarités. Cette méthode, en revanche,

généralise très difficilement, d'autant qu'elle repose sur une technique *ad hoc* d'identification des paramètres.

Afin de gagner en généricité tout en conservant une méthode de transfert intelligible et efficace à partir de peu d'exemples, nous nous sommes tourné vers l'analogie formelle, méthode de raisonnement à partir de cas. L'analogie permet d'obtenir des réponses avec une précision très élevée, mais un silence élevé également. Nous avons envisagé deux options pour réduire le silence : l'extension de la base d'exemples par génération analogique, et la relaxation des contraintes imposées par l'analogie formelle sur les séquences. Chacune permet une amélioration modeste, tout en dégradant la précision de manière variable.

Par ailleurs, l'utilisation du transfert analogique direct permet d'identifier et d'éditer implicitement et de manière générique les paramètres des requêtes. Le transfert analogique indirect permet quant à lui de traiter des requêtes nouvelles, c'est-à-dire des requêtes qui ne sont pas sur le modèle de requêtes préexistantes dans la base d'exemples.

Le transfert par analogie indirect repose sur le parcours de la base d'exemples afin de constituer des quadruplets de requêtes. Malgré l'utilisation d'une méthode d'indexation introduite par Langlais et Yvon fondée sur les arbres de comptage, cette étape reste coûteuse. Les techniques d'extension de la base et de relaxation des contraintes que nous avons proposées contribuent malheureusement à alourdir le coût de calcul des solutions. Nous avons exploré quelques pistes pour la réduction du temps de traitement en passant par une segmentation informée des séquences (voir le bilan de la deuxième partie, p. 139), sans résultats probants à ce jour. De nombreuses voies restent cependant ouvertes dans cette direction, nous en discutons dans les perspectives, section 9.2.

Enfin, nous avons également proposé un protocole expérimental pour le test du système de transfert en condition incrémentale. Il s'agit de simuler l'enseignement de l'utilisateur expert par l'ajout progressif des solutions aux requêtes soumises. Nous avons ainsi pu mesurer la courbe d'apprentissage du système, qui a notamment montré une stabilisation de la précision à son taux final (environ 74%) après environ 11 exemples de chaque commande, c'est-à-dire 11 reformulations de la requête associée.

9.1.3 Vue d'ensemble des résultats

L'intérêt particulier des résultats obtenus tient au faible nombre d'exemples qui ont été nécessaires pour les obtenir, ainsi qu'au fait que la base ne contient initialement aucun exemple. Ils démontrent l'efficacité du modèle choisi de transfert de langage par raisonnement analogique formel pour ces conditions.

Le cas général d'utilisation d'un tel assistant opérationnel incrémental ne contraint pas toujours à amorcer l'apprentissage à partir d'une base vide, quoique de nombreux domaines spécifiques sont particulièrement peu dotés. Cependant, à l'image d'une grande majorité des tâches d'apprentissage artificiel, il n'est pas réaliste de considérer à un quelconque moment dans l'utilisation du système que la base d'exemple contient, sous forme explicite ou implicite, l'intégralité des connaissances nécessaires à un comportement optimal vis-à-vis des utilisateurs. Pour cette raison, il nous semble primordial de concevoir ledit système de manière à conserver sa base d'exemples ouverte aux révisions, telles que l'ajout incrémental d'exemples que nous avons abordé, mais aussi leur retrait ou leur édition, qui peuvent être souhaitables afin de conserver une base représentative et à jour.

Par ailleurs, la précision en moyenne élevée obtenue à l'issue de nos expériences justifie l'emploi du raisonnement par analogie pour ce type de tâche d'interaction. Dans ce type de contexte, une approche générant plus de réponses au prix de multiples erreurs réduirait fortement l'utilité d'un tel assistant auprès de l'utilisateur expert, qui y perdrait du temps ainsi que sa sécurité auprès d'un utilisateur novice, qui ne saurait alors bien déceler ces erreurs.

9.2 Perspectives

Les résultats encourageants obtenus au cours de ce travail conduisent comme nous l'avons vu à de nombreuses questions et défis, tant sur le plan de l'exploration de l'approche analogique pour l'apprentissage incrémental, que sur le plan de l'exploitation des modèles proposés dans ce document. Nous donnons ici un aperçu des pistes qui nous semblent parmi les plus prometteuses.

9.2.1 Exploration

Comme nous l'avons évoqué à l'occasion de l'analyse des résultats obtenus, le raisonnement par analogie formelle présente encore des difficultés d'application à des bases d'exemples de taille conséquente (au delà de quelques dizaines de milliers d'exemples dans notre cas, jusqu'à au delà de quelques millions sur d'autres domaines). Ces difficultés proviennent d'un verrou scientifique majeur lié à toutes les méthodes de calcul d'analogies formelles existantes : leur complexité algorithmique combinatoire.

Des optimisations et heuristiques ont été proposées dans la littérature (*cf.* 4.4). Suivant la ligne directrice fixée en amont de ce travail, nous avons évité autant que possible les heuristiques, dont l'élagage de l'espace de recherche est susceptible d'exclure des solutions. Par ailleurs, de nouvelles optimisations sont envisageables à différents niveaux afin d'endiguer l'explosion combinatoire des algorithmes tout en considérant toujours l'intégralité des solutions existantes.

Relaxation raffinée

La relaxation des contraintes analogiques, telle que décrite dans ce travail, présente le désavantage outre la durée d'exécution, d'augmenter le nombre de réponses incorrectes produites par le système. D'après nos analyses, ces réponses incorrectes proviennent essentiellement de transferts indésirables entre langue naturelle et langage formel. Une modification envisageable à court terme pour éviter ce type d'erreurs serait de restreindre à nouveau les approximations analogiques autorisées afin de ne les permettre qu'entre des séquences du même domaine, c'est-à-dire, langue naturelle ou langage formel.

Par ailleurs, la segmentation dynamique présentée au bilan de la deuxième partie permettant un découpage plus intelligent des énoncés rendrait réalisable le remplacement d'expressions à un seul token par des expressions à plusieurs tokens (exemple : "le répertoire *courant*" $\rightarrow$ "le répertoire *dans lequel on est actuellement positionné*"). Ces remplacements ont en effet un coût prohibitif étant donné une segmentation non discriminante au niveau des mots, car une unité de déviation est ajoutée pour chaque mot supplémentaire.

Enfin, une piste intéressante concernant la relaxation lors de la recherche d'analogies dans l'espace d'entrée, c'est-à-dire les requêtes, consiste à demander une confirmation à l'utilisateur avant de lui soumettre une réponse approchée. En effet, le quadruplet presque proportionnel formé par recherche approchée d'analogies est souvent [1] composé d'au moins une équation analogique ayant une solution exacte. Dans ce cas, la comparaison de la solution exacte avec la requête approchée par l'utilisateur permet de s'assurer *a minima* que l'utilisateur valide leur équivalence sémantique, ce qui légitime donc – ou non le cas échéant – son utilisation pour une approximation.

Extension du contexte

Comme nous l'avons décrit dans la section 8.2, la segmentation dynamique doit permettre de réduire le temps d'exécution non pas par la manipulation de la base ni de l'algorithme lui-même, mais en ajustant le niveau de granularité auquel les séquences sont prises en compte par les algorithmes de vérification et de résolution analogique. Le découpage intelligent d'un énoncé selon des unités

1. Cette piste a été présentée de manière plus détaillée en section 6.5.

plus grandes permettrait de limiter la longueur des séquences, et rendrait donc possible le traitement d'énoncés plus longs. De plus, dans le cas d'énoncés représentant des requêtes, il serait intéressant d'enrichir la structure manipulée par l'analogie par des éléments pertinents du contexte comme la commande précédente, le contexte actuel d'exécution (architecture de la machine, répertoire de travail et son contenu). Tous ces éléments ne seront cependant pas toujours pertinents, ce qui pose problème lorsque l'on considère la résolution non relâchée d'analogies. Une alternative à la relaxation dont ce cas illustre bien l'intérêt serait la détermination par analogie de l'ensemble des éléments pertinents pour une résolution appropriée. Par extension, cette approche pourrait également compléter ou remplacer la relaxation telle que nous l'avons envisagée au chapitre 6.

Sélection des exemples de la base

La complexité algorithmique des approches que nous avons présentées rend problématique l'utilisation de bases d'exemples de grande taille. Habituellement, c'est sur des bases de grandes tailles que sont attendus les meilleurs résultats en termes de précision et de taux d'erreur.

Cependant, l'intérêt d'une grande base d'exemples n'est pas tant lié au nombre d'exemples qu'à la quantité d'information qu'elle contient. La corrélation de ces deux grandeurs n'est pas linéaire dans le cas général. On cherchera donc à augmenter la quantité d'information contenue dans la base d'exemples utilisée en limitant autant que possible sa taille. En un mot, on cherche à augmenter la densité d'information, ou entropie, de la base d'exemples.

Pour ce faire, il convient de sélectionner les exemples à y ajouter parmi les nouveaux cas corrects découverts, c'est-à-dire validés ou enseignés par l'utilisateur. Les exemples présentant une redondance avec la base existante ne seront pas ajoutés à la base active, mais à l'ensemble des exemples rencontrés et non utilisés lors de la recherche. Dans le cadre de l'analogie formelle, on peut définir un nouvel exemple comme redondant par rapport à la base s'il existe un sous-ensemble d'exemples de la base permettant un transfert analogique direct ou indirect de sa requête vers sa commande. Autrement dit, un exemple est redondant s'il fait déjà partie de la fermeture transitive analogique de la base, toutes méthodes de transfert confondues.

Par ailleurs, réduire la base d'exemples à son noyau analogique peut être envisagé en étendant tout d'abord la base par analogie, avant de sélectionner parmi l'ensemble d'exemples résultant le plus petit sous-ensemble permettant de générer l'intégralité de la base par analogie. La base d'exemples obtenue de cette manière a donc une entropie maximale.

Notons que si le coût calculatoire du parcours d'une base d'exemples trop grande est bien évité grâce à cette approche, celui du calcul récursif des analogies de la fermeture transitive, en revanche, devient nécessaire pour couvrir autant de cas que lorsque tous les nouveaux exemples sont ajoutés.

9.2.2 Exploitation

Les modèles que nous avons proposés pour le transfert de langages et l'apprentissage incrémental par interaction peuvent également être réutilisés en l'état afin d'étudier leur adéquation selon les paires de langages et les groupes d'utilisateurs. Les questions auxquelles cette application permettrait d'apporter des réponses sont : l'efficacité du raisonnement par analogie formelle varie-t-elle selon les langages utilisés ou les domaines d'intérêt du groupe des utilisateurs ? Si oui, dans quelle mesure et quelles sont les caractéristiques qui causent ces variations ?

Nous avons vu que les modèles proposés supportent mal de grandes bases de connaissances, cependant lors d'un apprentissage incrémental, la base de connaissances n'atteint que la taille strictement nécessaire pour les utilisateurs. Comment évolue cette taille au cours du temps, ou plutôt du nombre d'interactions entre l'assistant opérationnel et les utilisateurs ? Et dans quelle mesure varie-t-elle selon le nombre d'utilisteurs ? Les résultats obtenus au chapitre 7 montrent un progrès croissant dans le

nombre de réponses correctes apportées par le système au cours de l'incrément. On s'attend donc à ce que l'augmentation de la base soit rapide (fréquent) initialement, puis diminue jusqu'à tendre vers 0.

Une application prometteuse pour un assistant opérationnel incrémental de ce genre consiste en la mise en place d'une base d'exemples centralisée par crowdsourcing. Le principe, comparable à celui des plateformes colaboratives actuelles telles que Wikipedia ou StackOverflow, est de mettre en commun les compétences des utilisateurs du système. Ainsi il suffit pour une commande donnée que quelques utilisateurs l'enseignent au système pour que le système la maîtrise et réponde correctement à ce besoin pour l'essentiel des utilisateurs ultérieurs. Une telle application est également un moyen de recueillir des informations les moins biaisées, car en situation, concernant l'utilisation du système assistant par les utilisateurs ainsi que son expression naturelle, ainsi que leur évolution au cours du temps.

9.2.3 Voies sur le long terme

Enfin, le modèle de l'assistant opérationnel interactif ouvre sur des pistes à plus long terme, dont les possibilités de modélisation ne sont pas encore définies de manière claire.

En premier lieu, et pour rappeler l'exemple présenté en introduction, on peut penser à l'intégration de la modalité orale au système. Il s'agit d'ajouter une couche de traitement pour la reconnaissance et la synthèse vocale en amont du modèle existant. L'intérêt d'utiliser l'analogie pour un système oral est que les relations analogiques de surface ne se limitent pas à la forme textuelle des énoncés, mais peuvent également se transférer à leur forme phonétique [Yvon, 1997, Yvon, 1999]. Après reconnaissance des phonèmes à partir du signal audio, leur transcription en syllabes et mots devient facultative, car le transfert de langages peut aussi bien s'opérer entre le phonétique et le langage formel.

Cependant, la modalité orale soulève d'autres problèmes dans le cadre d'un assistant opérationnel : celui du nommage des paramètres. On conviendra en premier lieu que l'énonciation orale et écrite ne sera généralement pas identique pour les mêmes requêtes. C'est particulièrement vrai à propos des paramètres, pour lesquels on devra utiliser une périphrase ou bien une épellation. Dans d'autres cas, la modalité orale devient même problématique comme avec la requête :
"Télécharge le fichier à l'adresse https ://ocsync.limsi.fr/index.php/s/BHBXwh51BlGC5Jl"

Une seconde voie qui nous paraît intéressante sur le long terme concerne le développement du dialogue du système. Le système que nous avons proposé, bien qu'il traite des requêtes au contenu arbitraire, présente des capacités dialogiques complètement scriptées et immuables. De la même manière que nos habitudes langagières sont calquées sur des expressions couramment utilisées par soi-même et par notre entourage, le raisonnement analogique pourrait enrichir le système d'une capacité d'ajustement plus précis des actes de dialogue qu'il génère, c'est-à-dire au delà du seul remplissage automatique de phrases à trous (notons que ce principe lui-même relève du raisonnement analogique).

Index

Bibliographie

[Achananuparp *et al.*, 2008] ACHANANUPARP, P., HU, X. et SHEN, X. (2008). The evaluation of sentence similarity measures. *In Data Warehousing and Knowledge Discovery*, pages 305–316. Springer.

[Albright et Hayes, 2003] ALBRIGHT, A. et HAYES, B. (2003). Rules vs. analogy in english past tenses : A computational/experimental study. *Cognition*, 90(2):119–161.

[Allen *et al.*, 2007] ALLEN, J., CHAMBERS, N., FERGUSON, G., GALESCU, L., JUNG, H., SWIFT, M. et TAYSOM, W. (2007). Plow : a collaborative task learning agent. *In Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, AAAI'07, pages 1514–1519. AAAI Press.

[Ameixa *et al.*, 2014] AMEIXA, D., COHEUR, L., FIALHO, P. et QUARESMA, P. (2014). Luke, I am your father : dealing with out-of-domain requests by using movies subtitles. *In Intelligent Virtual Agents*, pages 13–21. Springer.

[Androutsopoulos *et al.*, 1995] ANDROUTSOPOULOS, I., RITCHIE, G. D. et THANISCH, P. (1995). Natural language interfaces to databases–an introduction. *Natural language engineering*, 1(01):29–81.

[Balbach et Zeugmann, 2009] BALBACH, F. J. et ZEUGMANN, T. (2009). Recent developments in algorithmic teaching. *In Language and Automata Theory and Applications*, pages 1–18. Springer.

[Banchs et Li, 2012] BANCHS, R. E. et LI, H. (2012). IRIS : a chat-oriented dialogue system based on the vector space model. *In Proceedings of the ACL 2012 System Demonstrations*, pages 37–42. Association for Computational Linguistics.

[Biber, 1993] BIBER, D. (1993). Representativeness in corpus design. *Literary and linguistic computing*, 8(4):243–257.

[Blaylock et Allen, 2004] BLAYLOCK, N. et ALLEN, J. F. (2004). Statistical goal parameter recognition. *In ICAPS*, volume 4, pages 297–304.

[Bobrow, 1964] BOBROW, D. G. (1964). *Natural language input for a computer problem solving system*. Thèse de doctorat.

[Boyé, 2016] BOYÉ, G. (2016). Pour une modélisation surfaciste de la flexion. le cas de la conjugaison du français. *In SHS Web of Conferences*, volume 27. EDP Sciences.

[Brown *et al.*, 1993] BROWN, P. F., PIETRA, V. J. D., PIETRA, S. A. D. et MERCER, R. L. (1993). The mathematics of statistical machine translation : Parameter estimation. *Computational linguistics*, 19(2):263–311.

[Cakmak et Thomaz, 2014] CAKMAK, M. et THOMAZ, A. L. (2014). Eliciting good teaching from humans for machine learners. *Artificial Intelligence*, 217:198–215.

[Cappeau et Gadet, 2007] CAPPEAU, P. et GADET, F. (2007). L'exploitation sociolinguistique des grands corpus.

[Chang et Lin, 2011] CHANG, C.-C. et LIN, C.-J. (2011). Libsvm : a library for support vector machines. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 2(3):27.

[Cohen, 1960] COHEN, J. (1960). A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46.

[Copestake et Jones, 1990] COPESTAKE, A. et JONES, K. S. (1990). Natural language interfaces to databases. *The Knowledge Engineering Review*, 5(04):225–249.

[Cornuéjols et Miclet, 2011] CORNUÉJOLS, A. et MICLET, L. (2011). *Apprentissage artificiel : concepts et algorithmes*. Editions Eyrolles.

[De Marneffe *et al.*, 2006] DE MARNEFFE, M.-C., MACCARTNEY, B., MANNING, C. D. *et al.* (2006). Generating typed dependency parses from phrase structure parses. *In Proceedings of LREC*, volume 6, pages 449–454.

[Devillers *et al.*, 2015] DEVILLERS, L., ROSSET, S., DUBUISSON DUPLESSIS, G., SEHILI, M. A., BÉCHADE, L., DELABORDE, A., GOSSART, C., LETARD, V., YANG, F., YEMEZ, Y., TÜRKER, B. B., SEZGIN, M., EL HADDAD, K., DUPONT, S., LUZZATI, D., ESTÈVE, Y., GILMARTIN, E. et NICK, C. (2015). Multimodal data collection of human-robot humorous interactions in the joker project. *In ACII*, Xi'an, China.

[Dijkstra, 1971] DIJKSTRA, E. W. (1971). *A short introduction to the art of programming*, volume 4. Technische Hogeschool Eindhoven Eindhoven.

[Dubuisson Duplessis *et al.*, 2017] DUBUISSON DUPLESSIS, G., CHARRAS, F., LETARD, V., LIGOZAT, A.-L. et ROSSET, S. (2017). Utterance retrieval based on recurrent surface text patterns. *In 39th European Conference on Information Retrieval (ECIR)*.

[Dubuisson Duplessis *et al.*, 2016] DUBUISSON DUPLESSIS, G., LETARD, V., LIGOZAT, A.-L. et ROSSET, S. (2016). Joker chatterbot – shared task chatbot description report. *In Workshop on Collecting and Generating Resources for Chatbots and Conversational Agents-Development and Evaluation*.

[Ducrot, 1998] DUCROT, O. (1998). *Dire et ne pas dire. Principes de sémantique linguistique*. Hermann, 3ème ed. édition.

[Duplessis *et al.*, 2016] DUPLESSIS, G. D., LETARD, V., LIGOZAT, A.-L. et ROSSET, S. (2016). Purely corpus-based automatic conversation authoring. *In 10th edition of the Language Resources and Evaluation Conference (LREC)*.

[Galliano *et al.*, 2005] GALLIANO, S., GEOFFROIS, E., MOSTEFA, D., CHOUKRI, K., françois BONASTRE, J. et GRAVIER, G. (2005). The ester phase ii evaluation campaign for the rich transcription of french broadcast news. *In in Proceedings of the 9th European Conference on Speech Communication and Technology (INTERSPEECH'05*, pages 1149–1152.

[Gentner, 1983] GENTNER, D. (1983). Structure-mapping : A theoretical framework for analogy. *Cognitive Science*, 7:155–170.

[Gentner *et al.*, 2001] GENTNER, D., HOLYOAK, K. J. et KOKINOV, B. N. (2001). *The analogical mind : Perspectives from cognitive science*. MIT press.

[Gilmartin et Campbell, 2016] GILMARTIN, E. et CAMPBELL, N., éditeurs (2016). *Just talking - casual talk among humans and machines, Worksphop LREC'16*.

[Giraud-Carrier, 2000] GIRAUD-CARRIER, C. (2000). A note on the utility of incremental learning. *AI Communications*, 13(4):215–223.

[Grice, 1975] GRICE, H. P. (1975). Logic and conversation. *In Syntax and Semantics : Vol. 3 : Speech Acts*, pages 41–58. Academic Press.

[Hesse, 1959] HESSE, M. B. (1959). On defining analogy. *In Proceedings of the Aristotelian Society*, volume 60, pages 79–100. JSTOR.

[Hofstadter *et al.*, 1994] HOFSTADTER, D. R., MITCHELL, M. *et al.* (1994). The copycat project : A model of mental fluidity and analogy-making. *Advances in connectionist and neural computation theory*, 2(31-112):29–30.

[Hofstadter et Sander, 2013] HOFSTADTER, D. R. et SANDER, E. (2013). *L'analogie au coeur de la pensée*. Odile Jacob.

[Jokinen et McTear, 2009] JOKINEN, K. et MCTEAR, M. (2009). Spoken dialogue systems. *Synthesis Lectures on Human Language Technologies*, 2(1):1–151.

[Jones et Galliers, 1996] JONES, K. S. et GALLIERS, J. R. (1996). *Evaluating natural language processing systems : An analysis and review*, volume 1083. Springer Science & Business Media.

[Juola et Baayen, 2005] JUOLA, P. et BAAYEN, R. H. (2005). A controlled-corpus experiment in authorship identification by cross-entropy. *Literary and Linguistic Computing*, 20(Suppl):59–67.

[Kaufmann et Bernstein, 2007] KAUFMANN, E. et BERNSTEIN, A. (2007). How useful are natural language interfaces to the semantic web for casual end-users ? *In The Semantic Web*, pages 281–294. Springer.

[Knuth, 1976] KNUTH, D. E. (1976). Mathematics and computer science : coping with finiteness. *Science (New York, NY)*, 194(4271):1235–1242.

[Koehn *et al.*, 2007] KOEHN, P., HOANG, H., BIRCH, A., CALLISON-BURCH, C., FEDERICO, M., BERTOLDI, N., COWAN, B., SHEN, W., MORAN, C., ZENS, R. *et al.* (2007). Moses : Open source toolkit for statistical machine translation. *In Proceedings of the 45th annual meeting of the ACL on interactive poster and demonstration sessions*, pages 177–180. Association for Computational Linguistics.

[Kollar *et al.*, 2013] KOLLAR, T., TELLEX, S., WALTER, M. R., HUANG, A., BACHRACH, A., HEMACHANDRA, S., BRUNSKILL, E., BANERJEE, A., ROY, D., TELLER, S. *et al.* (2013). Generalized grounding graphs : A probabilistic framework for understanding grounded language. *JAIR*.

[Kosub, 2016] KOSUB, S. (2016). A note on the triangle inequality for the jaccard distance. *arXiv preprint arXiv :1612.02696*.

[Kushman et Barzilay, 2013] KUSHMAN, N. et BARZILAY, R. (2013). Using semantic unification to generate regular expressions from natural language. *In Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics*. North American Chapter of the Association for Computational Linguistics (NAACL).

[Lafferty *et al.*, 2001] LAFFERTY, J., MCCALLUM, A. et PEREIRA, F. (2001). Conditional random fields : Probabilistic models for segmenting and labeling sequence data. *In Proceedings of the eighteenth international conference on machine learning, ICML*, volume 1, pages 282–289.

[Lamel *et al.*, 2000] LAMEL, L., ROSSET, S., GAUVAIN, J.-L., BENNACEF, S., GARNIER-RIZET, M. et PROUTS, B. (2000). The LIMSI ARISE System. *Speech Communication*, 31(4):339–354.

[Langlais et Patry, 2007] LANGLAIS, P. et PATRY, A. (2007). Translating unknown words by analogical learning. *In EMNLP-CoNLL*, pages 877–886.

[Langlais et Yvon, 2008] LANGLAIS, P. et YVON, F. (2008). Scaling up analogical learning. *In COLING (Posters)*, pages 51–54.

[Langlais *et al.*, 2009] LANGLAIS, P., YVON, F. et ZWEIGENBAUM, P. (2009). Improvements in analogical learning : application to translating multi-terms of the medical domain. *In Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 487–495. Association for Computational Linguistics.

[Lee *et al.*, 2009] LEE, C., JUNG, S., KIM, S. et LEE, G. G. (2009). Example-based dialog modeling for practical multi-domain dialog system. *Speech Communication*, 51(5):466–484.

[Leech, 2006] LEECH, G. (2006). New resources, or just better old ones ? the holy grail of representativeness. *Language and Computers*, 59(1):133–149.

[Lepage, 1998] LEPAGE, Y. (1998). Solving analogies on words : An algorithm. *In Proceedings of the 36th Annual Meeting of the Association for Computational Linguistics and 17th International Conference on Computational Linguistics - Volume 1*, ACL '98, pages 728–734, Stroudsburg, PA, USA. Association for Computational Linguistics.

[Lepage, 2004] LEPAGE, Y. (2004). Analogy and formal languages. *Electronic Notes in Theoretical Computer Science*, 53:180 – 191.

[Lepage et Denoual, 2005] LEPAGE, Y. et DENOUAL, E. (2005). Purest ever example-based machine translation : Detailed presentation and assessment. *Machine Translation*, 19(3-4):251–282.

[Letard *et al.*, 2015] LETARD, V., ROSSET, S. et ILLOUZ, G. (2015). Analogical reasoning for natural to formal language transfer. *In IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2015)*, page 8, Vietri sul Mare, Italy.

[Mandelbrot, 1965] MANDELBROT, B. (1965). Information theory and psycholinguistics. *BB Wolman and E.*

[Manning *et al.*, 2008a] MANNING, C. D., RAGHAVAN, P. et SCHÜTZE, H. (2008a). *Introduction to information retrieval*, volume 1. Cambridge university press Cambridge.

[Manning *et al.*, 2008b] MANNING, C. D., RAGHAVAN, P. et SCHÜTZE, H. (2008b). Scoring, term weighting and the vector space model. *Introduction to information retrieval*, 100:2–4.

[Marshall, 2002] MARSHALL, J. B. (2002). Metacat : a program that judges creative analogies in a microworld. *In Proceedings to the Second Workshop on Creative Systems*. Citeseer.

[Martinet, 1956] MARTINET, A. (1956). *Économie des changements phonétiques. Traité de phonologie diachronique*.

[Miclet *et al.*, 2008] MICLET, L., BAYOUDH, S. et DELHAY, A. (2008). Analogical dissimilarity : definition, algorithms and two experiments in machine learning. *Journal of Artificial Intelligence Research*, pages 793–824.

[Mogensen, 2009] MOGENSEN, T. Æ. (2009). *Basics of Compiler Design*.

[Nagao, 1984] NAGAO, M. (1984). A framework of a mechanical translation between japanese and english by analogy principle. *Artificial and human intelligence*, pages 351–354.

[Nio *et al.*, 2014] NIO, L., SAKTI, S., NEUBIG, G., TODA, T., ADRIANI, M. et NAKAMURA, S. (2014). Developing non-goal dialog system based on examples of drama television. *In Natural Interaction with Robots, Knowbots and Smartphones*, pages 355–361. Springer.

[Papineni *et al.*, 2002] PAPINENI, K., ROUKOS, S., WARD, T. et ZHU, W.-J. (2002). Bleu : a method for automatic evaluation of machine translation. *In Proceedings of the 40th annual meeting on association for computational linguistics*, pages 311–318. Association for Computational Linguistics.

[Popescu *et al.*, 2003] POPESCU, A.-M., ETZIONI, O. et KAUTZ, H. (2003). Towards a theory of natural language interfaces to databases. *In Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157. ACM.

[Prade et Richard, 2014] PRADE, H. et RICHARD, G. (2014). A short introduction to computational trends in analogical reasoning. *In Computational Approaches to Analogical Reasoning : Current Trends*, pages 1–22. Springer.

[Rapp, 2014] RAPP, R. (2014). Using word familiarities and word associations to measure corpus representativeness. *In Proceedings of the Ninth International Conference on Language Resources and Evaluation (LREC 2014)*.

[Rich et Waters, 1993] RICH, C. et WATERS, R. C. (1993). Approaches to automatic programming. *Advances in computers*, 37:1–57.

[Rosset *et al.*, 2006a] ROSSET, S., GALIBERT, O., ILLOUZ, G. et MAX, A. (2006a). Integrating spoken dialog and question answering : the ritel project. *In INTERSPEECH*.

[Rosset *et al.*, 2006b] ROSSET, S., GALIBERT, O., ILLOUZ, G. et MAX, A. (2006b). Integrating Spoken Dialog and Question Answering : the Ritel Project. *In InterSpeech'06*, Pittsburgh, USA.

[Saussure, 1916] SAUSSURE, F. d. (1916). *Cours de linguistique générale*. Bally, Charles and Sechehaye, Albert.

[Schatzmann *et al.*, 2006] SCHATZMANN, J., WEILHAMMER, K., STUTTLE, M. et YOUNG, S. (2006). A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The knowledge engineering review*, 21(2):97–126.

[Shortliffe *et al.*, 1975] SHORTLIFFE, E. H., DAVIS, R., AXLINE, S. G., BUCHANAN, B. G., GREEN, C. C. et COHEN, S. N. (1975). Computer-based consultations in clinical therapeutics : explanation and rule acquisition capabilities of the mycin system. *Computers and biomedical research*, 8(4): 303–320.

[Singh *et al.*, 2002] SINGH, S., LITMAN, D., KEARNS, M. et WALKER, M. (2002). Optimizing dialogue management with reinforcement learning : Experiments with the njfun system. *Journal of Artificial Intelligence Research*, 16:105–133.

[Somers *et al.*, 2009] SOMERS, H., DANDAPAT, S. et NASKAR, S. K. (2009). A review of ebmt using proportional analogies. *EBMT 2009 - 3rd Workshop on Example-Based Machine Translation*.

[Stroppa, 2005] STROPPA, N. (2005). *Analogy-Based Models for Natural Language Learning*. Phd thesis, Télécom ParisTech.

[Stroppa et Yvon, 2004] STROPPA, N. et YVON, F. (2004). Analogies dans les séquences : un solveur à états finis. *In Actes de la 11e Conférence Annuelle sur le Traitement Automatique des Langues Naturelles (TALN 2004)*.

[Stroppa et Yvon, 2006] STROPPA, N. et YVON, F. (2006). Du quatrième de proportion comme principe inductif : une proposition et son application à l'apprentissage de la morphologie. *Traitement automatique des langues*, 47(1).

[Stroppa et Yvon, 2007] STROPPA, N. et YVON, F. (2007). Formal models of analogical proportions. Rapport technique, École Nationale Supérieure des Télécommunications, Paris, France.

[Tadić *et al.*, 2012] TADIĆ, M., BEKAVAC, B., AGIĆ, Z., SREBAČIĆ, M., BEROVIĆ, D. et MERKLER, D. (2012). Early machine translation based semantic annotation prototype. Rapport technique D3.3.1, XLike project, www.xlike.org.

[Tellex *et al.*, 2011] TELLEX, S. A., KOLLAR, T. F., DICKERSON, S. R., WALTER, M. R., BANERJEE, A., TELLER, S. et ROY, N. (2011). Understanding natural language commands for robotic navigation and mobile manipulation.

[Tellier, 2005] TELLIER, I. (2005). *Modéliser l'acquisition de la syntaxe du langage naturel via l'hypothése de la primauté du sens*. Thèse de doctorat, Université Charles de Gaulle-Lille III.

[Turing, 1950] TURING, A. M. (1950). Computing machinery and intelligence. *Mind*, 59(236):433–460.

[Ukkonen, 1995] UKKONEN, E. (1995). On-line construction of suffix trees. *Algorithmica*, 14(3): 249–260.

[Vadas et Curran, 2005] VADAS, D. et CURRAN, J. R. (2005). Programming with unrestricted natural language. *In Proceedings of the Australasian Language Technology Workshop*, pages 191–199. Citeseer.

[van Schooten *et al.*, 2007] van SCHOOTEN, B. W., ROSSET, S., GALIBERT, O., MAX, A., ILLOUZ, G. *et al.* (2007). Handling speech input in the ritel qa dialogue system.

[Volkova *et al.*, 2013] VOLKOVA, S., CHOUDHURY, P., QUIRK, C., DOLAN, B. et ZETTLEMOYER, L. S. (2013). Lightly supervised learning of procedural dialog systems. *In ACL (1)*, pages 1669–1679. Citeseer.

[Weizenbaum, 1966] WEIZENBAUM, J. (1966). Eliza—a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45.

[Winograd, 1972] WINOGRAD, T. (1972). *Understanding natural language*. Thèse de doctorat.

[Young, 2006] YOUNG, S. (2006). Using pomdps for dialog management. *In in Proceedings of the 1st IEEE/ACL Workshop on Spoken Language Technologies (SLT'06*.

[Yu *et al.*, 1984] YU, V., FAGAN, L., BENNETT, S., CLANCEY, W., SCOTT, A., HANNIGAN, J., BLUM, R., BUCHANAN, B. et COHEN, S. (1984). An evaluation of mycin's advice. *Rule-based expert systems : The MYCIN experiments of the Stanford heuristic programming project*, pages 589–596.

[Yvon, 1997] YVON, F. (1997). Paradigmatic cascades : a linguistically sound model of pronunciation by analogy. *In Proceedings of the 35th Annual Meeting of the Association for Computational Linguistics and Eighth Conference of the European Chapter of the Association for Computational Linguistics*, pages 428–435. Association for Computational Linguistics.

[Yvon, 1999] YVON, F. (1999). Pronouncing unknown words using multi-dimensional analogies. *In EUROSPEECH*.

[Zipf, 1935] ZIPF, G. K. (1935). *The psycho-biology of language.*

ANNEXES

Annexe A

Liste des publications

A.1 Publications en lien avec la thèse

[1] Vincent Letard. Interaction homme-machine en domaine large à l'aide du langage naturel : une amorce par mise en correspondance. In *Rencontres des Etudiants Chercheurs en Informatique pour le Traitement Automatique des Langues (RECITAL 2014)*, Actes de Recital 2014, pages 81–91, Marseille, France, 01/07 au 04/07 2014.

[2] Vincent Letard, Gabriel Illouz, and Sophie Rosset. Reducing noise sensitivity of formal analogical reasoning applied to language transfer. In *Computational Analogy Workshop at ICCBR (ICCBR-CA 2016)*, The Twenty-Forth International Conference on Case-Based Reasoning - Workshop Proceedings, page 11, Atlanta, United-States (GA), 31/10 2016. Alexandra Coman & Stelios Kapetanakis.

[3] Vincent Letard, Gabriel Illouz, and Sophie Rosset. Évaluation de l'apprentissage incrémental par analogie. In *Conférence sur le Traitement Automatique des Langues Naturelles (TALN 2016)*, page 13p, Paris, France, 04/07 au 08/07 2016.

[4] Vincent Letard, Sophie Rosset, and Gabriel Illouz. A mapping-based approach for general formal human computer interaction using natural language. In *Annual Meeting of the Association for Computational Linguistics (ACL 2014)*, Proceedings of the ACL 2014 Student Research Workshop, pages 34–40, Baltimore, Maryland, USA, 22/06 au 27/06 2014.

[5] Vincent Letard, Sophie Rosset, and Gabriel Illouz. Analogical reasoning for natural to formal language transfer. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2015)*, page 8, Vietri sul Mare, Italy, 09/11 au 11/11 2015.

[6] Vincent Letard, Sophie Rosset, and Gabriel Illouz. Incremental learning from scratch using analogical reasoning. In *IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2016)*, 2016 IEEE 28th International Conference on Tools with Artificial Intelligence Proceedings, page 8, San Jose, United States (CA), 06/11 au 08/11 2016. Nikolaos Bourbakis, Anna Esposito, Amol Mali, and Miltos Alamaniotis.

A.2 Autres publications

[7] Franck Charras, Guillaume Dubuisson Duplessis, Vincent Letard, Anne-Laure Ligozat, and Sophie Rosset. Comparing system-response retrieval models for open-domain and casual conversational agent. In *Second Workshop on Chatbots and Conversational Agent Technologies (WOCHAT 2016)*, page 12, Los Angeles, USA, 20/09 2016.

[8] Franck Charras, Guillaume Dubuisson Duplessis, Vincent Letard, Anne-Laure Ligozat, and Sophie Rosset. Un système automatique de sélection de réponse en domaine ouvert intégrable à un système de dialogue social. In *Conférence sur le Traitement Automatique des Langues Naturelles (TALN 2016)*, page 3p, Paris, France, 04/07 au 08/07 2016.

[9] Laurence Devillers, Sophie Rosset, Guillaume Dubuisson Duplessis, Mohamed El Amine Sehili, Lucile Béchade, Agnès Delaborde, Clément Gossart, Vincent Letard, Fan Yang, Yucel Yemez, Bekir Turker, Metin T. Sezgin, Kévin El Haddad, Stéphane Dupont, Daniel Luzzati, Yannick Estève, Emer Gilmartin, and Nick Campbell. Multimodal data collection of human-robot humorous interactions in the joker project. In *International Conference on Affective Computing and Intelligent Interaction (ACII 2015)*, pages 348–354, Xi'an, China, 21/09 au 24/09 2015.

[10] Guillaume Dubuisson Duplessis, Lucile Béchade, Mohamed El Amine Sehili, Agnès Delaborde, Vincent Letard, Anne-Laure Ligozat, Paul Deléglise, Yannick Estève, Sophie Rosset, and Laurence Devillers. Nao is doing humour in the chist-era joker project. In *Annual Conference of the International Speech Communication Association*, pages 1072–1073, Dresde, Allemagne, 06/09 au 10/09 2015.

[11] Guillaume Dubuisson Duplessis, Vincent Letard, Anne-Laure Ligozat, and Sophie Rosset. Joker chatterbot. In *Computational Analogy Workshop at ICCBR (ICCBR-CA 2016)*, page 2, Portoroz, Slovenia, 28/05 2016.

[12] Guillaume Dubuisson Duplessis, Vincent Letard, Anne-Laure Ligozat, and Sophie Rosset. Purely corpus-based automatic conversation authoring. In *International Conference on Language Resources and Evaluation*, page 8, Portoroz, Slovenia, 23/05 au 28/05 2016. Nicoletta Calzolari (Conference Chair) and Khalid Choukri and Thierry Declerck and Marko Grobelnik and Bente Maegaard and Joseph Mariani and Asuncion Moreno and Jan Odijk and Stelios Piperidis.

Annexe B

Guide de collecte d'associations

Ce guide de collecte a été soumis aux participants pour constituer le corpus `bash2`. Version originale à cette adresse : https://perso.limsi.fr/letard/collecte.php

Consignes

J'ai besoin de petits ensembles de commandes extraits de votre historique `bash` à intervalles réguliers (1, 2 ou 3 jours) avec les requêtes correspondantes en français. Vous pouvez utiliser cette commande pour récupérer un lot aléatoire de commandes :

```
cat -n ~/.bash_history | sort -R | cut -f 2 -d"$(echo -e '\t')" | head -n 15
```

Écrire les requêtes en français

Écrivez pour chaque commande une requête correspondante en une ligne. La seule consigne pour cela est : **La requête doit vous permettre, ou à quelqu'un d'autre, de retrouver la commande sans ambiguïté.** Hormis cette condition importante, la rédaction est libre (comprendre naturelle).

Commandes à exclure

- les commandes inexactes (erreurs de syntaxe) :

  ```
  echo "Il manque le guillemet de fermeture !
  ```

- les programmes ou scripts non standard (ne provenant pas d'un paquet officiel), ou alias :

  ```
  ./mon_classifieur --stdin
  ```

  ```
  connect_to_cluster
  ```

- les commandes trop difficiles à écrire en une seule ligne en français :

  ```
  for i in `find . -name "*.cfg" -exec grep -l "Total Processes" {} \;`;
  do OFILE=$i.bak; NFILE=$i; mv $i{,.bak}; START=`grep -n "Total Processes"
  $OFILE -B2 | grep "-" | awk 'BEGIN{FS="-";} {print $1}' | head -n 1`;
  END=`grep -n "Total Processes" $OFILE -A4 | grep "-" | tail -n 1 |
  sed 's/\-\}//'`; python -c "lines=open('$OFILE').readlines()[0:$START-1];
  print ''.join(lines);lines=open('$OFILE').readlines()[$END:];
  print ''.join(lines)" > $NFILE; sed -i 's/\n\n\n/\n/g' $NFILE; done;
  ```

Commandes à include

Les doublons sont bienvenus, associez-y la requête qui vous vient à l'esprit à ce moment.

Format d'envoi

Stockez les commandes et requêtes dans le format le plus simple : le fichier texte. Un fichier texte contiendra les commandes, un autre les requêtes, dans le même ordre bien entendu et une ligne pour chacune.

Quelques exemples

#	`commandes.txt`	`requetes.txt`
1	`ls -lh`	Donne moi la liste détaillée des fichiers du répertoire courant, au format humain.
2	`cut -f 3-5 -d "," mon_fichier.csv`	Affiche les colonnes 3 à 5 du fichier csv mon_fichier.csv
3	`find src/ -iname "main*"`	Trouve-moi les fichiers qui commencent par "main" récursivement dans le répertoire src/
4	`who`	fais-moi un who
5	`who`	Qui est connecté sur la machine ?

Droits du participant

Les commandes de votre historique `bash` que vous choisissez d'envoyer ou d'écarter sont laissées à votre discrétion. Selon le cas, vous êtes libre d'anonymiser une commande sensible, ou même de l'omettre dans votre envoi.

Exemple d'anonymisation :

`curl -data "id=mon_id" https://url.sensible.org/page.cgi`	Envoie l'id mon_id à https ://url.sensible.org/page.cgi
`curl -data "arg=anon1" https://anon2`	Envoie l'arg anon1 à https ://anon2

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Apprentissage incrémental de modèles de domaines par interaction dialogique

Mots-clés : apprentissage incrémental, systèmes interactifs, apprentissage symbolique

Résumé : L'intelligence artificielle est la discipline de recherche d'imitation ou de remplacement de fonctions cognitives humaines. À ce titre, l'une de ses branches s'inscrit dans l'automatisation progressive du processus de programmation. Il s'agit alors de transférer de l'intelligence ou, à défaut de définition, de transférer de la charge cognitive depuis l'humain vers le système, qu'il soit autonome ou guidé par l'utilisateur. Dans le cadre de cette thèse, nous considérons les conditions de l'évolution depuis un système guidé par son utilisateur vers un système autonome, en nous appuyant sur une autre branche de l'intelligence artificielle : l'apprentissage artificiel. Notre cadre applicatif est celui de la conception d'un assistant opérationnel incrémental, c'est-à-dire d'un système capable de réagir à des requêtes formulées par l'utilisateur en adoptant les actions appropriées, et capable d'apprendre à le faire. Pour nos travaux, les requêtes sont exprimées en français, et les actions sont désignées par les commandes correspondantes dans un langage de programmation. L'apprentissage du système est effectué à l'aide d'un ensemble d'exemples constitué par les utilisateurs eux-mêmes lors de leurs interactions. Nous avons collecté plusieurs ensembles d'exemples pour l'évaluation des méthodes d'apprentissage, en analysant et réduisant progressivement les biais induits. Le protocole que nous proposons est fondé sur l'amorçage incrémental des connaissances du système à partir d'un ensemble vide ou très restreint. Nous utilisons donc une méthode de raisonnement à partir de cas : le raisonnement par analogie formelle. Nous montrons que cette méthode permet une précision très élevée dans les réponses du système, mais également une couverture relativement faible. L'extension de la base d'exemples par analogie est explorée afin d'augmenter la couverture des réponses données. Dans une autre perspective, nous explorons également la piste de rendre l'analogie plus tolérante au bruit et aux faibles différences en entrée en autorisant les approximations, ce qui a également pour effet la production de réponses incorrectes plus nombreuses. La durée d'exécution de l'approche par analogie, déjà de l'ordre de la seconde, souffre beaucoup de l'extension de la base et de l'approximation. Nous avons exploré plusieurs méthodes de segmentation des séquences en entrée afin de réduire cette durée, mais elle reste encore le principal obstacle à contourner pour l'utilisation de l'analogie formelle dans le traitement automatique de la langue. Enfin, l'assistant opérationnel incrémental fondé sur le raisonnement analogique a été testé en condition incrémentale simulée, afin d'étudier la progression de l'apprentissage du système au cours du temps. On en retient que le modèle permet d'atteindre un taux de réponse stable après une dizaine d'exemples vus en moyenne pour chaque type de commande. Bien que la performance effective varie selon le nombre total de commandes considérées, cette propriété ouvre sur des applications intéressantes dans le cadre incrémental du transfert depuis un domaine riche (la langue naturelle) vers un domaine moins riche (le langage de programmation).

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title : Incremental Learning of Domain Models by Dialogic Interaction

Keywords : incremental learning, interactive systems, symbolic machine learning

Abstract : Artificial Intelligence is the field of research aiming at mimicking or replacing human cognitive abilities. As such, one of its subfields is focused on the progressive automation of the programming process. In other words, the goal is to transfer cognitive load from the human to the system, whether it be autonomous or guided by the user. In this thesis, we investigate the conditions for making a user-guided system autonomous using another subfield of Artificial Intelligence : Machine Learning. As an implementation framework, we chose the design of an incremental operational assistant, that is a system able to react to natural language requests from the user with relevant actions. The system must also be able to learn the correct reactions, incrementally. In our work, the requests are in written French, and the associated actions are represented by corresponding instructions in a programming language (here R and bash). The learning is performed using a set of examples composed by the users themselves while interacting. Thus they progressively define the most relevant actions for each request, making the system more autonomous. We collected several example sets for evaluation of the learning methods, analyzing and reducing the inherent collection biases. The proposed protocol is based on incremental bootstrapping of the system, starting from an empty or limited knowledge base. As a result of this choice, the obtained knowledge base reflects the user needs, the downside being that the overall number of examples is limited. To avoid this problem, after assessing a baseline method, we apply a case base reasoning approach to the request to command transfer problem: formal analogical reasoning. We show that this method yields answers with a very high precision, but also a relatively low coverage. We explore the analogical extension of the example base in order to increase the coverage of the provided answers. We also assess the relaxation of analogical constraints for an increased tolerance of analogical reasoning to noise in the examples. The running delay of the simple analogical approach is already around 1 second, and is badly influenced by both the automatic extension of the base and the relaxation of the constraints. We explored several segmentation strategies on the input examples in order to reduce reduce this time. The delay however remains the main obstacle to using analogical reasoning for natural language processing with usual volumes of data. Finally, the incremental operational assistant based on analogical reasoning was tested in simulated incremental condition in order to assess the learning behavior over time. The system reaches a stable correct answer rate after a dozen examples given in average for each command type. Although the effective performance depends on the total number of accounted commands, this observation opens interesting applicative tracks for the considered task of transferring from a rich source domain (natural language) to a less rich target domain (programming language).

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France