Les méthodes « apprendre à chercher » en traitement automatique des langues : un état de l'art

Elena Knyazeva — Guillaume Wisniewski — François Yvon

LIMSI, CNRS, Univ. Paris-Sud, Université Paris-Saclay,
Campus universitaire bât 508, Rue John von Neumann
F - 91405 Orsay cedex
{knyazeva,wisniews,yvon}@limsi.fr

RÉSUMÉ. *L'apprentissage structuré est au fondement des méthodes modernes d'apprentissage automatique pour le traitement automatique des langues (TAL). Dans cet article, nous étudions une famille d'algorithmes d'apprentissage structuré, les algorithmes de la famille « apprendre à chercher », qui diffèrent fondamentalement des méthodes classiques telles que les champs markoviens aléatoires, et permettent donc de mettre en évidence certains des compromis de l'apprentissage structuré en TAL. Nous présentons également un panorama des applications de ces techniques en TAL, en discutant les bénéfices découlant de leur utilisation.*

ABSTRACT. *Structured prediction lies at the heart of modern Natural language Processing (NLP). In this paper, we study a specific family of structured learning algorithms, loosely refered to as "Learning-to-search" algorithms. They differ in several important ways from more studied methods such as Conditional Random Fields, and their study highlights several important trade-offs of structured learning for NLP. We also present an overview of existing applications of these techniques to NLP problems and discuss their potential benefits.*

MOTS-CLÉS : *traitement automatique des langues, apprentissage structuré, apprendre à chercher.*

KEYWORDS: *Natural Language Processing, Structured learning, Learning to search.*

1. Introduction

L'utilisation de méthodes d'apprentissage supervisé est l'approche de référence en traitement automatique des langues (TAL), et permet d'obtenir les meilleures performances aussi bien sur des tâches *intermédiaires* d'analyse linguistique (étiquetage en parties du discours, analyse syntaxique, identification de relations de coréférence, détection d'entités nommées, etc.), que pour des tâches *finalisées* (traduction automatique, dialogue, extraction d'information, etc.).

La mise en œuvre de ces méthodes repose sur la modélisation, sur un corpus d'apprentissage, de corrélations statistiques entre des descripteurs et une ou plusieurs variables d'intérêt, permettant ensuite le calcul (l'inférence) des valeurs de ces variables lorsqu'elles ne sont pas observées. Cette stratégie est rendue particulièrement difficile en TAL par (a) le besoin de modéliser des phénomènes à partir d'observations parfois très éparses ; (b) le caractère *structuré* des entités linguistiques (mots, phrases, documents), qui implique des dépendances souvent complexes, et potentiellement à longue distance, entre les variables que l'on doit prédire. Un exemple typique de (b) est l'accord grammatical entre un sujet et le verbe principal, qui contraint les deux termes à partager des traits communs (personne et nombre en français) indépendamment de leurs positions respectives dans une phrase, qui peuvent être arbitrairement éloignées. Prendre en compte ces dépendances demande d'employer des techniques *d'apprentissage structuré* (Bakir *et al.*, 2007 ; Smith, 2011), qui sont donc les techniques de référence pour de nombreuses tâches : *parsing* stochastique, alignement de mots, détection de coréférence, extraction d'information, traduction automatique, etc.

Toute méthode d'apprentissage structuré doit faire face aux questions (a) et (b). Une réponse pragmatique consiste à ne prendre en compte que des dépendances « proches »[1], par exemple en bornant *a priori* les dépendances au voisinage immédiat selon une hypothèse markovienne à l'ordre 1 ou 2. Utiliser des hypothèses simplificatrices rend l'estimation moins sensible au caractère épars des données. Cela permet également l'utilisation de méthodes d'estimation et d'inférence reposant sur *des algorithmes exacts* de complexité raisonnable tels que l'algorithme de Viterbi (1967) pour l'inférence dans les modèles de Markov cachés.

Une alternative consiste à utiliser des contextes plus larges, en renonçant à l'exactitude de l'inférence, remplacée pour la circonstance par des techniques de *recherche gloutonne*. C'est le cas des méthodes « apprendre à chercher »[2] (L2S[3]) (Daumé *et al.*, 2009 ; Ross *et al.*, 2011) que nous étudions ici, avec pour ambition d'en donner une présentation compacte et précise. Ces méthodes sont intéressantes car, outre leurs performances souvent équivalentes à celles des méthodes standard comme les champs markoviens conditionnels (Tellier et Tommasi, 2011), elles jettent un nouvel éclai-

1. « Proche » n'est pas nécessairement entendu en relation avec l'ordre linéaire des mots et peut aussi s'interpréter au travers les relations de domination dans un arbre.

2. La terminologie est un peu fluctuante et l'on parle également d'apprentissage par imitation. Nous nous en tenons à « apprendre à chercher », qui nous semble moins ambigu.

3. Pour *Learning to Search*.

rage sur les compromis de l'apprentissage structuré en TAL. L'idée principale de ces techniques consiste à (a) reformuler l'inférence d'une structure linguistique sous la forme d'une séquence de décisions locales, chaque séquence de décisions définissant une trajectoire dans l'espace (combinatoirement grand) des structures possibles ; (b) apprendre à réaliser ces décisions locales en utilisant *un classifieur sensible aux coûts* (c'est-à-dire un classifieur qui prend en compte le fait que les différentes erreurs de classification peuvent impliquer différents coûts (Elkan, 2001)) qui intègre toutes les décisions prises dans le passé. L'enjeu est alors de minimiser les erreurs du classifieur local, afin de maximiser les chances de construire une structure entièrement correcte. Il importe donc que les exemples d'apprentissage du classifieur soient le plus proches possibles des contextes locaux qui seront effectivement visités pendant l'inférence ; le choix optimal ne consistant pas nécessairement à utiliser ceux qui découlent des annotations de référence.

Cet article est organisé comme suit : nous posons, à la section 2 un certain nombre de concepts nécessaires à la bonne exposition des principales méthodes d'apprendre à chercher de la littérature, en particulier les notions de *politique expert et tuteur* ; nous détaillons ensuite, à la section 3, les deux principales familles d'algorithmes (AggreVaTe et SEARN) de la littérature, qui reposent toutes les deux sur la notion *d'itérations de politique*. Au terme de ces deux sections théoriques, la section 4 présente un certain nombre d'applications au TAL de ces techniques, en essayant de mettre en évidence leur intérêt par rapport à l'état de l'art. Nous concluons à la section 5 en présentant quelques problèmes ouverts et pistes de développement pour ces techniques. Une annexe, disponible en ligne, détaille les preuves esquissées dans le corps du texte[4].

2. Apprendre à chercher : un autre point de vue sur l'apprentissage structuré

2.1. *Apprentissage structuré : quelques définitions*

2.1.1. *Apprentissage structuré*

Dans cette section, nous posons les principales notions nécessaires à la compréhension des méthodes détaillées dans la section 3. Comme pour tout problème d'apprentissage supervisé, nous observons un ensemble de couples associant une entrée $x \in X$ et une sortie de référence $y \in Y$, supposés échantillonnés sous une distribution $\widetilde{D}$ inconnue. Pour fixer les idées, nous utiliserons en guise d'illustration la tâche de traduction. Pour cette tâche, x et y sont respectivement la phrase à traduire et sa traduction de référence. Nous supposons également donnée une fonction de perte $L(y, \hat{y})$, qui évalue l'erreur commise lorsque l'on prédit $\hat{y}$ au lieu de la référence y. La métrique standard pour mesurer la qualité d'une traduction est le score BLEU (Papineni *et al.*, 2002)[5] ; la fonction de perte peut être, par exemple, l'opposé de ce score.

4. `perso.limsi.fr/yvon/publications/sources/Knyazeva19L2S-annexe.pdf`

5. BLEU n'est pas une métrique bien adaptée à l'évaluation de phrases isolées, et il faut plutôt utiliser des variantes de ce score.

Le point de vue standard en apprentissage structuré (Smith, 2011) considère un ensemble paramétrique de fonctions f_θ permettant d'associer x à $y = f_\theta(x)$ et d'optimiser les paramètres θ en minimisant la perte moyenne sur un ensemble d'apprentissage. Le point de vue des méthodes « apprendre à chercher » est différent : le focus est mis sur le processus de construction de la sortie y, envisagé à travers l'exploration de l'espace de toutes les sorties possibles. Pour apprendre à bien prédire y, il importe de savoir explorer l'espace de recherche et d'y prendre les bonnes décisions. Cet objectif peut sembler moins directement lié à la résolution du problème initial, mais il a l'avantage de se formaliser comme une suite de décisions simples, qu'il est possible d'apprendre à réaliser avec des outils génériques et peu coûteux de classification. Nous introduisons quelques notions de base, avant de formaliser la relation entre le problème structuré et la séquence de problèmes de classification associée.

2.1.2. *Espace de recherche*

Nous considérons des problèmes d'apprentissage *structuré*, pour lesquels l'ensemble des sorties y possibles pour un x donné est un ensemble combinatoire, représenté par un graphe. Formellement, pour une entrée x et sa sortie de référence $y \in Y$, l'espace de recherche $G(x, y) = \langle S,\, A,\, S_f,\, s_i,\, r\rangle$ est composé de[6] :

– S est l'ensemble (fini) des états possibles ;

– A est l'ensemble (fini) des actions possibles ;

– $S_f \subset S$ est l'ensemble des états finaux ;

– $s_i \in S$ est un état initial (unique pour chaque espace de recherche) ;

– $r(s, a)$ est la *récompense* associée à l'action a dans l'état s.

La figure 1 représente un espace de recherche pour la tâche de traduction. Chaque état correspond à une traduction partielle produite ; une action consiste à choisir un ou plusieurs mots pour continuer la traduction partielle. L'état initial correspond à la traduction vide, et toute traduction complète est un état final. La définition des récompenses sera explicitée ci-dessous.

On notera A_s l'ensemble des actions réalisables dans l'état s et $s' = s \circ a$ l'opération d'extension qui sélectionne l'action $a \in A_s$ dans l'état s afin d'atteindre l'état s'. Dans le cadre de ce travail, cette fonction est supposée déterministe, ce qui signifie que la connaissance de l'état et de l'action est suffisante pour déterminer l'état d'arrivée. L'espace de recherche forme un graphe sans cycle à l'exception d'une boucle sur chaque état final, dont la fonction sera clarifiée dans la section 2.2.1[7].

On appelle alors *chemin de longueur* t une suite $\alpha = (s_1, a_1) \dots (s_t, a_t)$ de couples dans $S \times A$, telle que $s_{t'+1} = s_{t'} \circ a_{t'}$ pour $1 \leq t' \leq t - 1$. Un chemin

6. Ceci est une simplification du concept de processus de décision markovien (Puterman, 1994), dans laquelle l'état d'arrivée est une fonction déterministe de l'état de départ et de l'action effectuée.

7. Un tel graphe est généralement appelé treillis, bien que, contrairement à la définition standard, il contienne plusieurs états finaux.

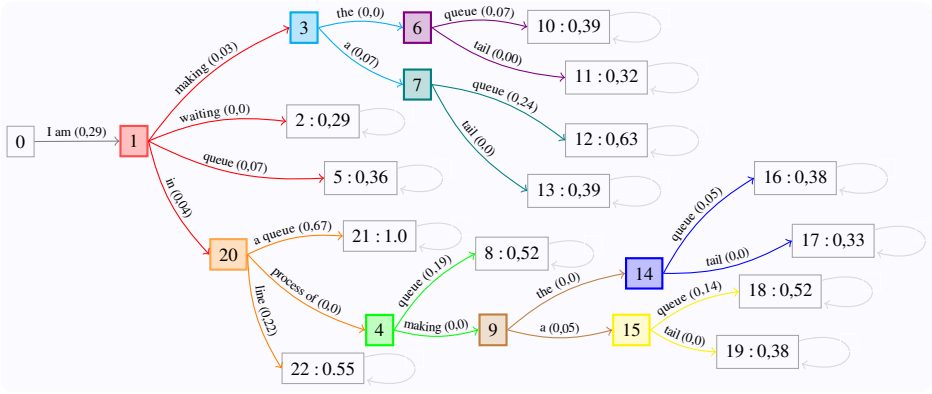

Figure 1. *Exemple d'un espace de recherche de traductions en anglais de la phrase « Je suis en train de faire la queue ». Les états pour lesquels plusieurs actions sont possibles sont colorés.*

est *complet* si $s_1 = s_i$ et $s_t \circ a_t \in S_f$; dans le cas contraire, on parlera d'un *chemin partiel*. La séquence d'actions $a_1 \dots a_t$ composant un chemin complet α peut être vue comme une suite d'instructions permettant de construire la sortie $\hat{y}_\alpha$ [8].

La *récompense cumulée* d'un chemin complet est la somme des récompenses associées aux actions le long de ce chemin :

$$R(\alpha) = \sum_{(s,a)\in\alpha} r(s,a),$$

et correspond à la qualité de la solution $\hat{y}_\alpha$. La récompense est définie en relation avec la fonction de perte de la manière suivante : la différence entre les récompenses cumulées de deux chemins est égale à l'opposé de la différence de pertes associées aux solutions correspondantes. Pour deux chemins α_1 et α_2, on a donc :

$$L(y, \hat{y}_{\alpha_1}) - L(y, \hat{y}_{\alpha_2}) = -(R(\alpha_1) - R(\alpha_2)).$$

Dans notre exemple, les états finaux sont équipés des scores 2-BLEU[9] des traductions produites ; ces scores correspondent aux récompenses cumulées le long du chemin menant dans les états finaux. Les récompenses locales sont définies de manière que la somme des récompenses sur chaque chemin complet soit égale au score de la traduction produite. Leurs valeurs sont données pour chaque action entre parenthèses.

8. Pour simplifier les démonstrations, on notera T la longueur minimale telle que pour chaque espace de recherche simulé sous $\widetilde{D}$, tous les chemins de longueur T ou plus soient complets.

9. On utilise généralement le score 4-BLEU, qui combine les précisions $1, 2, 3$, et 4-grammes (Papineni *et al.*, 2002). Nous utilisons ici 2-BLEU, qui est mieux adapté à l'évaluation de phrases isolées, pour lesquelles les précisions d'ordre supérieur sont souvent nulles.

2.1.3. *Politique et notions associées*

Une *politique* est une fonction déterminant l'action à choisir dans un état donné. Cette fonction peut être déterministe ou stochastique. Dans le premier cas, on parle de *politique déterministe* $\tau : S \mapsto A$, associant à chaque état $s \in S$ une unique action $a \in A_s$. L'application de la politique τ depuis l'état initial jusqu'à un état final produit un chemin α_τ, qui correspond à la sortie $\hat{y}_{\alpha_\tau}$. Un exemple de politique déterministe pour l'espace de recherche de la figure 1 est :

$$\check{\tau} = \{1 \to 5, 3 \to 7, 4 \to 8, 6 \to 10, 7 \to 12, 9 \to 15, 14 \to 16, 15 \to 18, 20 \to 4\}$$

Cet ensemble correspond aux choix des actions pour chaque état où plusieurs actions sont possibles. L'application de $\check{\tau}$ depuis l'état initial produit la traduction « I am queue ». $\check{\tau}$ peut aussi être appliquée depuis un état interne : son application depuis l'état 9 produit la traduction « I am in process of making a queue ».

Une politique stochastique est une fonction stochastique $\pi : S \mapsto A$ qui, à chaque état $s \in S$, associe une action selon une certaine distribution de probabilité sur A_s. Dans ce cas, le chemin produit, ainsi que la sortie structurée, sont également stochastiques. Nous noterons la distribution correspondante sur les chemins par $\widetilde{\alpha}_\pi$. Les politiques déterministes étant des cas particuliers de politiques stochastiques, on supposera dans la suite que, sauf mention explicite, toute politique π est stochastique.

2.1.3.1. Qualité de la politique

La qualité de la politique π se mesure à l'aune de la qualité de la sortie structurée qu'elle produit. Afin de mettre en évidence cette relation, nous utiliserons la même notation pour exprimer ces deux notions. Ainsi, la qualité d'une politique déterministe est simplement la perte associée à la sortie structurée produite par cette politique :

$$L(\tau) = L(y, \hat{y}_{\alpha_\tau}).$$

La qualité d'une politique stochastique est la perte moyenne sur les sorties produites :

$$L(\pi) = \mathbb{E}_{\alpha \sim \widetilde{\alpha}_\pi}[L(y, \hat{y}_\alpha)].$$

L'objectif de l'apprentissage est de trouver, à partir des données étiquetées, une politique qui minimise cette perte.

2.1.3.2. Utilisation de politique

Quelques notions supplémentaires pour manipuler les politiques sont nécessaires :

– $s \circ \pi^t$ est le résultat de l'application d'une suite de t actions choisies par la politique π depuis l'état s ; en particulier, $s \circ \pi = s \circ \pi^1$ pour une seule application de la politique. La politique π étant supposée stochastique, ce résultat l'est aussi (c'est une distribution sur les états pouvant être obtenus). *C'est le cas de tous les objets définis dans cette section lorsqu'une politique stochastique est utilisée ;*

– $s \circ \pi^*$ note le résultat de l'application d'un nombre arbitraire (entre 0 et T) d'actions choisies par la politique π dans l'état s.

Ces notions peuvent être généralisées au cas d'une distribution sur les états de départ $\widetilde{s}$. Par exemple, $\widetilde{s} \circ \pi$ est la distribution sur les états obtenue en appliquant l'action choisie par π sur la distribution $\widetilde{s}$ d'états de départ. On introduit maintenant des notions concernant les chemins produits à l'aide d'une politique π :

– $s \xrightarrow{\pi, t} \cdot$ est le chemin de t actions construit à l'aide de π à partir de l'état s ;

– $s \xrightarrow{\pi, \cdot} S_f$ est le chemin construit à l'aide de la politique π à partir de l'état s jusqu'à un état final (la longueur du chemin n'est pas fixée).

On peut également combiner plusieurs politiques pour construire un chemin complexe. $s \xrightarrow{\pi_1, t_1} \cdot \xrightarrow{\pi_2, t_2} \cdot \xrightarrow{\pi_3, \cdot} S_f$, est ainsi un chemin où les t_1 premières actions sont choisies par la politique π_1, les t_2 actions suivantes sont choisies par la politique π_2 et la suite du chemin jusqu'à l'état final est construite par la politique π_3.

Ces notions se généralisent aussi au cas d'une distribution d'états de départ.

2.1.3.3. Récompenses cumulées et fonctions de valeur

Nous étendons maintenant les notions de récompenses aux chemins. La *récompense cumulée* $R(\widetilde{\alpha})$ d'un chemin $\widetilde{\alpha}$ stochastique se définit par :

$$R(\widetilde{\alpha}) = \mathbb{E}_{\alpha \sim \widetilde{\alpha}}[R(\alpha)].$$

La *fonction de valeur* V *d'un état* s vis-à-vis de la politique π est définie par :

$$V(s; \pi) = R(s \xrightarrow{\pi, \cdot} S_f).$$

Elle représente la récompense cumulée depuis l'état s jusqu'à l'état final, lorsque toutes les actions sont choisies selon la politique π. Pour notre exemple, $V(0; \check{\tau}) = 0,36$ (ce qui correspond au score de la traduction « I am queue ») et $V(1; \check{\tau}) = 0,07$ (ce qui correspond au score de la même traduction moins la récompense associée à l'action effectuée pour arriver dans l'état 1).

La *fonction de valeur* Q *d'un couple* (*état* s, *action* $a \in A_s$) vis-à-vis de la politique π représente la récompense cumulée depuis l'état s jusqu'à l'état final sachant que la première action est a et que les actions suivantes sont choisies selon π :

$$Q(s, a; \pi) = R(s \xrightarrow{\cdot} s \circ a \xrightarrow{\pi, \cdot} S_f).$$

Dans notre exemple, $Q(1, 1 \rightarrow 20; \check{\tau}) = 0,23$ (le score de la traduction « I am in process of queue » moins la récompense de la première action $0 \rightarrow 1$).

Il est possible d'étendre ces fonctions au cas où l'état et l'action sont stochastiques, le cas important étant celui où l'action a est donnée par une politique stochastique. Soient π_1 et π_2 deux politiques quelconques, s distribué selon $\tilde{s}$. On a alors :

$$V(\tilde{s}; \pi_2) = \mathbb{E}_{s \sim \tilde{s}} \left[R(s \xrightarrow{\pi_2, \cdot} S_f) \right] \quad Q(\tilde{s}, \pi_1; \pi_2) = \mathbb{E}_{s \sim \tilde{s}} \left[R(s \xrightarrow{\pi_1, 1} \cdot \xrightarrow{\pi_2, \cdot} S_f) \right]$$

2.1.4. *L'expert et le tuteur*

L'*expert* π_{exp} est une politique qui permet, lorsqu'elle est exécutée de bout en bout, de construire la meilleure solution présente dans l'espace de recherche[10]. Dans notre cas, cette meilleure solution est la traduction de référence « I am in a queue ». Soit $\alpha^*(G)$ l'ensemble des chemins de l'espace de recherche G, alors pour la politique expert la propriété suivante est vérifiée :

$$\hat{y}_{\pi_{\text{exp}}} = \underset{\alpha \in \alpha^*(G)}{\arg\min}\ L(y, \hat{y}_\alpha).$$

Dans ce travail, on suppose avoir accès à un expert plus puissant qui, *pour toute solution partielle*, permet de construire la meilleure solution complète possible[11]. C'est-à-dire que même si le début d'un chemin est construit à l'aide d'une autre politique, l'expert permet de trouver la meilleure manière de le poursuivre. Avec les notations introduites dans la section précédente, cet expert peut être exprimé par[12] :

$$\pi_{\text{exp}}(s) = \underset{a \in A_s}{\arg\max}\, Q(s, a, \pi_{\text{exp}}).$$

Ainsi, l'expert est la politique choisissant l'action qui conduira à la meilleure récompense cumulée, sachant que toutes les actions suivantes jusqu'à l'état final seront également choisies par la politique expert. Pour l'exemple de la figure 1 :

$$\check{\tau}_{\text{exp}} = \{1 \to 20, 3 \to 7, 4 \to 8, 6 \to 10, 7 \to 12, 9 \to 15, 14 \to 16, 15 \to 18, 20 \to 21\}.$$

Cette politique suivie du début à la fin construit la traduction de référence « I am in a queue » ; appliquée depuis l'état 4, elle construit la meilleure traduction atteignable « I am in process of making a queue ». Certains algorithmes présentés dans ce travail nécessitent un expert capable d'agir de la meilleure manière possible étant donné non seulement le début du chemin, mais en sachant aussi que la suite du chemin sera construite par une autre politique. Un tel expert est appelé un *tuteur*. La politique tuteur peut être formalisée par :

$$\pi_{\text{tut}}(s, \pi') = \underset{a \in A_s}{\arg\max}\, Q(s, a; \pi').$$

Ainsi, le tuteur choisit l'action impliquant la meilleure récompense cumulée sachant que toutes les actions suivantes jusqu'à l'état final seront choisies par la politique π'. La politique tuteur pour l'exemple figure 1 vis-à-vis de la politique $\check{\tau}$ est :

$$\check{\tau}_{\text{tut}}(\cdot, \check{\tau}) = \{1 \to 3, 3 \to 7, 4 \to 8, 6 \to 10, 7 \to 12, 9 \to 15, 14 \to 16, 15 \to 18, 20 \to 21\}$$

10. Cette meilleure solution n'est pas nécessairement identique à la référence, puisqu'il est possible que celle-ci ne soit pas présente dans l'espace de recherche. Ce problème est documenté (par exemple) dans (Liang *et al.*, 2006 ; Sokolov *et al.*, 2013).

11. Cette idée rejoint la notion d'« oracle complet » présentée dans (Goldberg et Nivre, 2012).

12. Si plusieurs actions permettent d'obtenir la valeur Q maximale, alors la politique expert est non déterministe. Dans ce travail nous ne considérons par spécialement ce cas ; plus de détails peuvent être trouvés par exemple dans (Knyazeva *et al.*, 2015).

Cette politique est différente de la politique expert $\check{\tau}_{\text{exp}}$ seulement pour l'état 1 : en proposant l'action $1 \rightarrow 3$ plutôt que l'action $1 \rightarrow 20$ elle tient compte du fait que les actions suivantes seront prises avec la politique $\check{\tau}$ et que forcer l'action $1 \rightarrow 3$ dans ce cas permettra d'obtenir le score final $0,63$ au lieu de $0,52$.

2.1.5. *La perte de potentiel*

Dans ce travail on suppose qu'il est possible de calculer la perte immédiate apportée par une action a (toujours vis-à-vis d'une politique). On appelle cette perte immédiate la *perte de potentiel* :

$$l(s, a, \pi) = \max_{a' \in A_s} Q(s, a'; \pi) - Q(s, a; \pi)$$

pour tout état $s \in S$ et pour toute action $a \in A_s$ vis-à-vis de la politique π. Pour notre exemple, $l(1, 1 \rightarrow 3, \check{\tau}_{\text{exp}}) = 0,37$, ce qui correspond à la différence des scores de « I am in a queue » et « I am making a queue ») et $l(1, 1 \rightarrow 3, \check{\tau}) = 0$ ($1 \rightarrow 3$ est la meilleure action dans l'état 1 quand le reste des actions est effectué selon $\check{\tau}$).

Comme pour les fonctions V et Q, nous généralisons la perte de potentiel aux cas où états et actions sont stochastiques. Quand l'action stochastique est donnée par une politique stochastique π_1 et l'état s est représenté par une distribution $\widetilde{s}$, on obtient :

$$l(\widetilde{s}, \pi_1, \pi_2) = \mathbb{E}_{s \sim \widetilde{s}} \left[\max_{a' \in A_s} Q(s, a'; \pi_2) - Q(s, \pi_1; \pi_2) \right].$$

2.2. *Réduction de l'apprentissage structuré à un problème de classification sensible aux coûts*

Intuitivement, apprendre à chercher consiste à apprendre, pour chaque état de l'espace de recherche, à choisir localement l'action qui conduira à la meilleure sortie possible. Nous formalisons ici ce lien entre classification locale et apprentissage structuré.

2.2.1. *Fonction objectif*

Comme expliqué ci-dessus, suivre la politique expert sur la trajectoire complète permet d'obtenir la meilleure solution présente dans l'espace de recherche. Il est néanmoins rarement possible d'imiter parfaitement la politique expert. À défaut, l'objectif de l'apprentissage est alors de trouver la politique π permettant de construire des sorties conduisant à la plus petite perte $L(\pi)$ possible.

Lemme 2.1 *La perte d'une politique π se décompose selon (la démonstration est donnée en annexe)*[13] *:*

$$L(\pi) = L(\pi_{\exp}) + T \cdot l(\widetilde{s_i} \circ \pi^*, \pi; \pi_{\exp}) \quad [1]$$

où $\widetilde{s_i}$ est une distribution sur les états initiaux.

Ce lemme exprime le fait que la perte totale se décompose en une perte associée à la politique expert $\pi_{\exp}$ (la référence n'étant pas toujours atteignable dans l'espace de recherche) et en des pertes de potentiel par rapport à la politique expert associées à chaque action réalisée par la politique π. La perte associée à la politique expert est fixe et ne dépend pas de la politique π. Pour minimiser la perte totale, il suffit donc de minimiser la perte moyenne de potentiel par rapport à la politique expert :

$$\arg\min_{\pi \in \Pi} L(\pi) = \arg\min_{\pi \in \Pi} l(\widetilde{s_i} \circ \pi^*, \pi; \pi_{\exp}). \quad [2]$$

Autrement dit, il s'agit d'apprendre une politique qui prend les meilleures décisions locales possible comparativement à celles de l'expert. Le problème global d'apprentissage d'une structure est ainsi réduit à une suite de problèmes de classification. Nous étudions ci-dessous les difficultés de l'optimisation de [2], ainsi que les méthodes approximatives de la littérature.

2.2.2. *Ensemble d'apprentissage*

Le problème [2] est difficile à résoudre en raison de son aspect circulaire : sa résolution demande de minimiser une perte moyenne *sur une certaine distribution d'états*, qu'il importe de caractériser ; or, calculer cette distribution présuppose de connaître la politique utilisée pour construire des trajectoires le long desquelles ces états sont visités. Il est possible de rendre ce problème simple en sélectionnant une distribution fixe (c'est-à-dire indépendante de la politique) d'états $\widetilde{s}_{\text{fix}}$ pour remplacer la distribution $\widetilde{s_i} \circ \pi^*$. On pourra ensuite minimiser $l(\widetilde{s}_{\text{fix}}, \pi; \pi_{\exp})$ par rapport à la politique π. Avec un choix raisonnable pour la distribution $\widetilde{s}_{\text{fix}}$, on espère que cette fonction approxime correctement la fonction objectif $l(\widetilde{s_i} \circ \pi^*, \pi; \pi_{\exp})$.

Une fois que les états sont fixés, le problème d'optimisation devient un problème standard de classification multiclasse sensible aux coûts. Nous appelons les triplets (état, actions accessibles, pertes associées) les *situations* auxquelles une politique est confrontée ; ils représentent l'ensemble des informations à partir desquelles la politique doit faire le choix d'une action. En supposant que l'on sait calculer cet ensemble de situations, toute méthode permettant d'apprendre un classifieur sensible aux coûts permet de trouver la politique π minimisant l'objectif $l(\widetilde{s}_{\text{fix}}, \pi; \pi_{\exp})$[14].

La figure 2.(a) représente graphiquement un ensemble d'apprentissage composé de deux situations auxquelles la politique est confrontée dans les états 1 et 20. L'abscisse et l'ordonnée de chaque point-action correspondent aux deux caractéristiques

13. Techniquement, la boucle présente sur chaque état final nous permet de prolonger chaque chemin jusqu'à la longueur T même si l'état final est atteint plus tôt ; cela nous permet de consi-

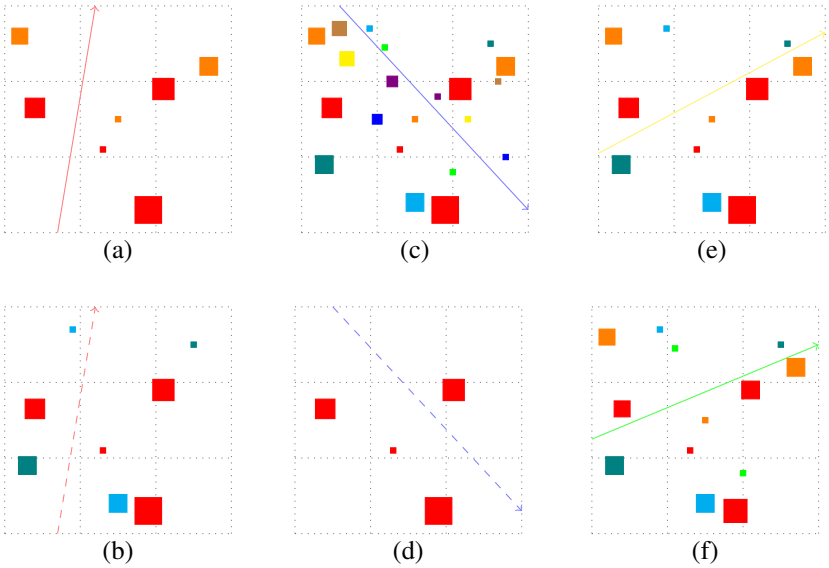

Figure 2. *Les ensembles d'apprentissage correspondant à l'application de différents algorithmes d'apprentissage à l'espace de recherche présenté sur la figure 1. La traduction de référence est « I am in a queue ». Chaque carré représente une action. L'abscisse et l'ordonnée de chaque point-action correspondent aux deux caractéristiques associées à cette action. Les couleurs correspondent aux couleurs des actions de la figure 1, tandis que les tailles des carrés correspondent (approximativement) à des pertes de potentiel associées. Parmi les points de même couleur, celle qui est la plus à gauche du séparateur correspond à l'action choisie. Les séparateurs continus correspondent à des classifieurs appris sur l'ensemble représenté, tandis que les séparateurs en pointillé sont donnés pour aider la visualisation et correspondent à des classifieurs qui doivent être appliqués sur les ensembles d'actions donnés.*

associées à cette action, et la taille des points représente (approximativement) la perte de potentiel[15] de l'action correspondant.

---

dérer uniquement les chemins de cette longueur. Pour des raisons de simplicité, nous utiliserons cette technique de manière systématique dans la suite.

14. Nous supposons dans cette section théorique que l'erreur liée à la taille limitée de l'ensemble d'apprentissage, ainsi qu'à l'estimation non exacte des coûts, est négligeable.

15. Si la politique par rapport à laquelle la perte de potentiel est calculée n'est pas indiquée on suppose par défaut que c'est la politique expert.

2.3. *Méthodes non itératives*

Nous commençons par étudier deux stratégies simples, dont dérivent deux instanciations possibles de $\widetilde{s}_{\text{fix}}$: (a) apprendre sur le chemin de l'expert (b) apprendre sur l'espace complet. L'étude de ces cas simples met clairement en évidence deux difficultés importantes qui se rencontrent également dans les approches plus complexes : l'*effet d'avalanche* et le *gaspillage de ressources*.

2.3.1. *Apprendre sur le chemin de l'expert : l'effet d'avalanche*

L'apprentissage sur le chemin de l'expert vise à imiter son comportement, en ne s'intéressant qu'aux situations qu'il rencontre, et donc à minimiser la perte suivante :

$$l(\widetilde{s}_i \circ \pi^*_{\text{exp}}, \pi; \pi_{\text{exp}}). \quad [3]$$

Cette approche est problématique car les seules situations vues à l'apprentissage sont des situations « idéales ». Une fois la politique apprise, toute erreur lors de son application mène dans un état (non optimal), qui peut différer des états vus à l'apprentissage. Cette différence entre les états considérés à l'apprentissage et ceux qui sont rencontrés à l'inférence augmente le risque que de nouvelles erreurs soient commises, ce qui peut déclencher une réaction en chaîne. Apprendre sur les seules trajectoires expertes expose donc à *l'effet d'avalanche* [16]. La perte moyenne de potentiel [3] obtenue en apprentissage donne alors une mauvaise estimation de la perte totale $L(\pi)$.

Une borne possible pour la perte totale est la suivante (Ross *et al.*, 2011) :

$$L(\pi) \leq L(\pi_{\text{exp}}) + p_{\text{err}} \cdot T \cdot L_{\max}, \quad [4]$$

où p_{err} est la probabilité de choisir une action différente de celle de l'expert dans un état *sur la trajectoire optimale*. Sur une trajectoire de longueur T, la probabilité de commettre une erreur est $p_{\text{err}} \cdot T$, et peut, au pire, conduire à une perte maximale $L_{\max}$, car même si une erreur n'implique qu'une perte de potentiel locale très petite, l'état d'arrivée de l'action choisie représente une situation inconnue pour la politique apprise, qui risque de conduire à de nouvelles erreurs. Cette garantie est insuffisante.

Les graphiques 2.(a) et 2.(b) illustrent cet effet d'avalanche. Comme le chemin de l'expert est $0 \rightarrow 1 \rightarrow 20 \rightarrow 21$, l'ensemble d'apprentissage sera composé des situations associées aux états 1 et 20 (dans les états 0 et 21, il n'y a pas de possibilité de choisir). Toutefois, avec une politique linéaire, les actions de l'expert ne peuvent pas être choisies, car ce chemin n'est pas linéairement séparable des autres. La politique apprise sur cet ensemble choisit alors les meilleures actions parmi les ensembles séparables. Le problème est que, en choisissant l'action $1 \rightarrow 3$, la politique doit réaliser des actions dans des états pour lesquels elle n'a pas été entraînée, et elle commet une erreur grave dans l'état 7, ce qui amène à une mauvaise traduction (0,39 BLEU).

16. Ce problème est aussi parfois appelé « *exposure bias* » (Ranzato *et al.*, 2016) ou encore « propagation d'erreur » (Lê et Fokkens, 2017) dans la littérature.

2.3.2. *Apprendre sur l'espace complet : le gaspillage de ressources*

Dans cette deuxième stratégie naïve, l'ensemble d'apprentissage comprend tous les états de l'espace de recherche. Soit $\widetilde{s}_{\text{uni}}$ la distribution uniforme sur l'ensemble des états. Supposons que l'on dispose d'un générateur de cette distribution, on peut alors, comme précédemment, résoudre un problème de classification sensible aux coûts pour trouver la politique qui minimise la perte de potentiel correspondante :

$$l(\widetilde{s}_{\text{uni}}, \pi; \pi_{\text{exp}}) \quad [5]$$

Inclure des exemples de toutes les situations pouvant se présenter à l'inférence soulève deux problèmes. D'abord un problème calculatoire, car l'espace de recherche pour une tâche structurée a généralement un nombre combinatoire d'états. Ensuite, même quand cette stratégie est techniquement réalisable, son efficacité n'est pas garantie, car la perte [5] ne reflète pas correctement la performance moyenne sur une trajectoire choisie par la politique π. Pour obtenir une garantie théorique, une possibilité est d'utiliser sa borne supérieure $\sup_{s\sim\widetilde{s}_{\text{uni}}}[l(s, \pi; \pi_{\text{exp}}]$ selon :

$$L(\pi) \leq L(\pi_{\text{exp}}) + \sup_{s\sim\widetilde{s}_{\text{uni}}}[l(s, \pi; \pi_{\text{exp}}] \cdot T \quad [6]$$

Intuitivement, le deuxième problème est lié au fait que, dans un cadre d'apprentissage automatique tel que le nôtre, les ressources du système sont limitées. Si tous les états possibles sont considérés lors de l'apprentissage, une partie des paramètres du classifieur sera consacrée à modéliser des phénomènes très éloignés du chemin suivi par l'expert et qui ne seront que très peu utiles lors de l'inférence.

Les figures 2.(c) et 2.(d) illustrent le gaspillage de ressources. L'ensemble de situations auxquelles la politique peut être confrontée est linéairement non séparable, le meilleur compromis étant le séparateur bleu figure 2.(c). Parmi les neuf situations « apprises », la politique associée à ce séparateur n'en rencontre qu'une seule pendant l'inférence. On voit figure 2.(d) que le séparateur choisi n'est pas optimal pour cette situation et amène à une mauvaise traduction (0,36 BLEU).

3. Apprendre à chercher : itération de politique

3.1. *Principe général*

La section précédente a montré que la composition de l'ensemble d'apprentissage de la politique a une grande importance : il est inutile de gaspiller des ressources en apprenant des situations que l'on ne rencontrera pas en pratique, tout comme il est risqué de se retrouver dans une situation que l'on n'a pas étudiée pendant l'apprentissage.

La notion *d'itération de politique* est à la base de toutes les méthodes d'apprentissage présentées ci-dessous. L'idée générale de ces méthodes, parfois également appelées *interactives* (Sun *et al.*, 2017), est d'apprendre une série de politiques, en utilisant les politiques déjà apprises pour construire un nouvel ensemble d'apprentissage et apprendre une nouvelle politique.

Utiliser une ou plusieurs politiques apprises antérieurement pour construire les nouvelles instances d'apprentissage permet de créer un ensemble d'états proches des conditions de l'inférence. Dans certaines situations, il est préférable d'utiliser également une politique apprise pour évaluer les coûts des actions. Cela fournira des coûts plus réalistes, car les pertes locales « expertes » sont fondées sur des anticipations souvent trop optimistes du futur (Chang *et al.*, 2015b). Deux politiques déterminent donc l'ensemble d'apprentissage : π_{in} la politique d'entrée (*roll-in policy*), et π_{out} est la politique de sortie (*roll-out*). On atteint les états qui constituent l'ensemble d'apprentissage en utilisant π_{in} ; on évalue la perte locale associée aux actions en supposant que la suite du calcul utilise π_{out}. On note l'ensemble d'apprentissage obtenu $\mathrm{Clf\_Prob}(\pi_{\mathrm{in}}, \pi_{\mathrm{out}})$. Ces bases posées, nous présentons ci-dessous deux familles d'algorithmes « apprendre à chercher », DAGGER et SEARN.

3.2. *La famille DAGGER*

Dans les algorithmes de la famille DAGGER[17], la politique est apprise sur l'accumulation de tous les exemples collectés depuis le début de la procédure d'entraînement. La philosophie de cette approche est d'apprendre à agir dans de nouvelles situations tout en préservant les connaissances accumulées lors des itérations précédentes. Asymptotiquement, cela permet de trouver une politique adaptée à l'ensemble des situations qu'elle peut potentiellement rencontrer. En revanche, l'ensemble d'apprentissage risque de contenir plus de situations que celles qui sont utiles en pratique, ce qui expose au problème du gaspillage de ressources.

3.2.1. *L'algorithme AggreVaTe*

L'algorithme AggreVate (Ross et Bagnell, 2014) part de la politique expert, et à chaque itération construit un nouvel ensemble d'apprentissage $\mathrm{Clf\_Prob}(\pi_{n-1}, \pi_{\mathrm{exp}})$ en utilisant la politique de l'itération précédente, les pertes associées aux actions étant évaluées par rapport à la politique expert. Cet ensemble est fusionné avec celui de l'itération précédente et une nouvelle politique est apprise en résolvant le problème de classification ainsi posé. La politique finale est choisie à l'aide de données de validation parmi les politiques apprises. Cette procédure correspond à l'algorithme 1[18].

Pour la classe de politiques convexes et en supposant un nombre infini d'itérations, on montre que les performances de la politique ainsi obtenue sont égales aux performances de la *meilleure* politique pour un ensemble de trajectoires accumulées au cours de l'apprentissage. Ce résultat est formalisé dans le théorème 3.1.

17. DAGGER (Ross *et al.*, 2011) est historiquement l'algorithme le plus connu de cette famille. Néanmoins, nous fondons notre présentation sur un algorithme plus récent, AggreVaTe (Ross et Bagnell, 2014) qui possède de meilleures garanties théoriques.

18. Dans cet article nous présentons une version légèrement simplifiée de AggreVaTe (qui permet toutefois d'obtenir les mêmes garanties théoriques) ; une comparaison avec la version originale se trouve en annexe.

Algorithme 1 : AggreVaTe : agrégation des données d'apprentissage

Données : données d'apprentissage et de validation $\sim \widetilde{D}$, ensemble paramétrique Π

Initialisation : $\pi_0 \leftarrow$ politique arbitraire de Π, $\mathcal{E} \leftarrow \emptyset$;
pour $n = 1..N$ **faire**
  Construire le problème d'apprentissage $\mathcal{E}_n = \mathrm{Clf\_Prob}(\pi_{n-1}, \pi_{\mathrm{exp}})$;
  Fusionner les exemples d'apprentissage $\mathcal{E} = \mathcal{E} \cup \mathcal{E}_n$;
  Apprendre π_n en minimisant la perte sur $\mathcal{E}$;
fin
En utilisant les données de validation, choisir π_{res} parmi $\{\pi_n\}_{n=1}^N$;
retourner π_{res};

Théorème 3.1 *Soit* Π *un ensemble paramétrique convexe de politiques,* $\breve{l}$ *un majorant convexe de la perte de potentiel*[19] *par rapport aux paramètres définissant* Π*. AggreVaTe construit une suite* $\{\pi_n\}_{n=1}^N$ *telle que pour au moins une politique* π_{res} *on a :*

$$L(\pi_{\mathrm{res}}) \leq L_{\mathrm{exp}} + T \cdot \min_{\pi \in \Pi} \frac{1}{N} \sum_{n=1}^{N} \breve{l}(\widetilde{s_i} \circ \pi_n, \pi, \pi_{\mathrm{exp}}) + O\left(\frac{1}{N}\right) \qquad [7]$$

La meilleure politique π_{res} *peut être sélectionnée à l'aide de l'ensemble de validation.*

Notons que la garantie obtenue n'est pas une garantie d'optimalité. Minimiser la perte $\sum_{n=1}^{N} \breve{l}(\widetilde{s_i} \circ \pi_n^*, \pi, \pi_{\mathrm{exp}})$ sur les trajectoires obtenues avec un ensemble de politiques plutôt que sur les seuls états obtenus avec la politique courante $\breve{l}(\widetilde{s_i} \circ \pi^*, \pi, \pi_{\mathrm{exp}})$ soulève le problème du gaspillage de ressources. En fait, rien ne garantit ni la qualité de l'ensemble de trajectoires considéré $\cup_{n=1}^{N} \widetilde{s_i} \circ \pi_n^*$ ni la bonne allocation des ressources d'apprentissage entre des trajectoires plus ou moins pertinentes.

Un exemple de déroulement de AggreVaTe est donné par les figures 2.(a) et 2.(e). Comme nous l'avons vu, le séparateur rouge de la figure 2.(a) a été choisi en prenant en compte seulement les états 1 et 20. À l'inférence, d'autres états sont rencontrés et ajoutés dans l'ensemble d'apprentissage. Le nouveau séparateur jaune (figure 2.(e)) trouve la deuxième meilleure traduction de l'espace de recherche avec le score 0,63. L'algorithme a convergé[20].

19. La démonstration de ce théorème repose sur la théorie de l'optimisation convexe en ligne où la fonction de perte est supposée être convexe. De manière générale, ce n'est pas le cas pour la perte de potentiel définie dans la section 2.1.4, c'est pourquoi un majorant convexe est utilisé ici (Ross et Bagnell, 2014). Notons que cela implique également une limitation sur la classe de politiques considérée qui est restreinte aux politiques convexes.

20. La convergence rapide est ici due à l'heureuse disposition des points sur le plan, un choix fait par simplicité. Dans le cas général, la convergence n'est pas garantie et la meilleure politique doit en général être choisie à l'aide d'un ensemble de validation (voir l'annexe pour plus de détails).

3.3. *La famille SEARN*

L'idée générale de SEARN (Daumé *et al.*, 2009) est d'apprendre progressivement une politique en partant de la politique expert et en confiant de plus en plus de décisions à la politique apprise, jusqu'à ce que cette politique ne fasse plus appel à l'expert. Dans cet article, nous présentons SEARN de manière légèrement différente de la version originale (Daumé *et al.*, 2009), ce qui simplifie les garanties théoriques associées. Les différences avec la version originale sont discutées en annexe.

Algorithme 2 : SEARN

Données : données d'apprentissage $\sim \widetilde{D}$, ensemble paramétrique Π, nombre d'itérations N, politique expert déterministe π_{exp}

Initialisation : $\pi_0 \leftarrow \pi_{\text{exp}}$;
pour $n \in 1..N$ **faire**
  Construire l'ensemble d'apprentissage $\mathcal{E}_n = \text{Clf\_Prob}(\pi_{n-1}, \pi_{n-1})$;
  Apprendre τ_n avec les exemples $\mathcal{E}_n$;
  $\pi_n = \text{Interpol}(\pi_{n-1}, \tau_n)$;
fin
retourner $\bar{\pi}_N$;

SEARN utilise la politique $\pi_n = \text{Interpol}(\pi_{n-1}, \tau_n)$ qui est une politique composite. Elle utilise le classifieur τ_n pour prendre une unique décision (aléatoire) sur une trajectoire et utilise la politique π_{n-1} pour les autres décisions. Ce qui donne pour π_n la formule récursive suivante :

$$\pi_n = \text{Interpol}(\pi_{n-1}, \tau_n) = \begin{cases} \tau_n & \text{pour } \hat{t} = \text{rand}(1, T) \\ \pi_{n-1} & \text{pour } t \neq \hat{t} \end{cases} \quad [8]$$

Pour formuler les garanties théoriques, nous introduisons une nouvelle notion de perte, calculée par rapport à l'action choisie par une autre politique. Nous avons besoin de comparer le cas où le chemin est obtenu en utilisant la politique π_1 à l'exception d'une position où la politique π_2 est utilisée, par rapport au cas où π_1 construit le chemin entier. Cela représente une perte liée à l'utilisation locale de π_2 :

$$l_{\text{loc}}(\pi_2; \pi_1) = V(\widetilde{s}_i; \pi_1) - Q(\widetilde{s}_i \circ \pi_1^*, \pi_2; \pi_1).$$

Les garanties théoriques de SEARN sont alors données par le théorème suivant :

Théorème 3.2 *Soit $\bar{\pi}_N$ la politique obtenue après N itérations de SEARN simple et après le remplacement des appels à l'expert par des décisions aléatoires. L'inégalité suivante est alors vérifiée*[21] *:*

$$L(\bar{\pi}_N) \leq L(\pi_{\text{exp}}) + \sum_{n=1}^{N} l_{\text{loc}}(\tau_n; \pi_{n-1}) + T \left(\frac{T-1}{T} \right)^N L_{\max} \quad [9]$$

21. Voir la démonstration complète en annexe.

où $L_{\max}$ (constante) est la perte maximale possible dans les espaces de recherche $\tilde{D}$.

Ce théorème signifie que la qualité de chaque nouvelle politique apprise est globalement déterminée par la somme des pertes *locales* obtenues à chaque itération. Les changements lents et contrôlés de la politique permettent d'assurer que l'effet d'avalanche est quasiment absent. Après N itérations, il existe encore une faible probabilité que la politique π_N fasse appel à l'expert, qui n'est pas disponible à l'inférence. On suppose que dans cette situation, la décision est prise aléatoirement, et implique la pire perte possible $L_{\max}$. Mais comme la probabilité de faire appel à l'expert diminue chaque itération, cela ne rajoute finalement qu'un petit terme supplémentaire dans la perte associée à la politique finale. Pour fixer les ordres de grandeur, $(T-1)^N T^{-(N-1)} \approx 2 \times 10^{-4}$ pour $T = 10$ et $N = 100$.

Un exemple de déroulement de SEARN est donné par les figures 2.(a) et 2.(f). Le séparateur rouge trouvé à la première itération (2.(a)) ne tient pas compte du fait que l'action $1 \rightarrow 3$, choisie dans l'état 1, amènera dans les états non explorés lors de l'apprentissage. Lors de la deuxième itération les états 3 et 7 rencontrés après avoir effectué cette action sont ajoutés dans l'ensemble d'apprentissage (ainsi que l'état 4 qui est à la fin d'un autre chemin exploré par SEARN)[22]. Le chemin $0 \rightarrow 1 \rightarrow 3 \rightarrow 7 \rightarrow 12$ choisi par le nouveau séparateur vert (2.(f)) est séparable et de bonne qualité ; cela termine l'algorithme. Notons que les pertes associées aux actions rouges sont changées, car ce ne sont plus les pertes de potentiel vis-à-vis de la politique expert, mais vis-à-vis de la politique composite qui choisit une action sur 6 (6 étant la longueur maximale du chemin) à l'aide de la politique « rouge » . Notamment, les pertes associées aux actions $1 \rightarrow 3$, $1 \rightarrow 2$, $1 \rightarrow 5$ sont diminuées par rapport à la perte de l'action de l'expert $1 \rightarrow 20$, car la politique composite a une probabilité non nulle de faire un mauvais choix dans l'état 20.

3.3.1. *Comparaison avec AggreVaTe*

Un point important concernant les garanties de SEARN est qu'elles ne donnent aucune information sur la qualité de la politique obtenue par rapport aux autres politiques de la classe considérée. On ne sait notamment pas si elle est proche de la politique optimale de cette classe ou non. Ces garanties ne nous informent que sur l'écart par rapport à l'expert. C'est une différence importante avec AggreVaTe qui garantit d'obtenir une politique presque optimale pour un certain nombre de problèmes, mais qui n'évite toutefois pas le gaspillage de ressources.

Notons également que, contrairement à AggreVaTe, pour obtenir les garanties [9], SEARN a besoin aussi bien des scores d'expert que des scores du tuteur, ce qui complique son utilisation en pratique. En revanche, SEARN ne met pas de limitation sur

22. Comme SEARN est un algorithme stochastique, pour donner une illustration précise, il faudrait ajouter les probabilités d'apparition de chaque situation dans l'ensemble d'apprentissage. Ici nous ne prenons pas en compte ce détail pour ne pas alourdir l'illustration, car dans tous les cas le chemin choisi par le séparateur sera le même (parce que ce chemin est séparable).

la classe de politiques considérée, alors que les garanties théoriques de AggreVate ne sont valables que pour les politiques convexes.

3.4. *Références supplémentaires*

Les sections précédentes nous ont confrontés au problème de la poule et de l'œuf : pour entraîner la politique optimale, il faut construire un bon problème d'apprentissage, mais pour construire celui-ci, la politique doit déjà être connue. Les méthodes L2S se ramènent alors à un problème d'optimisation de la répartition des ressources afin d'obtenir la politique optimale, contrairement à *l'apprentissage par renforcement* ou *par imitation*, qui font face à des difficultés supplémentaires d'exploration.

Le vocabulaire utilisé dans cet article (politique, fonction de valeur, récompense...) est en fait largement emprunté à l'apprentissage par renforcement (Sutton et Barto, 1998), tandis que l'utilisation que nous faisons de l'expert provient de l'apprentissage par imitation ; néanmoins, la nature du problème considéré ici est relativement différente. Dans le cadre L2S, nous disposons d'un accès direct aux pertes de potentiel pour chaque action, contrairement à l'apprentissage par renforcement où il est en général possible de n'observer que des informations partielles sur la récompense cumulée finale et où l'accès au choix de l'expert n'est jamais envisagé. De même, la disponibilité d'un simulateur puissant permettant d'obtenir les pertes de potentiel nous éloigne également de l'apprentissage par imitation où généralement seules des traces du comportement de l'expert sont disponibles (Geist, 2016).

De nombreuses stratégies présentées dans cette section sont en fait bien connues dans le cadre de l'apprentissage par renforcement. Par exemple, le problème de gaspillage de ressources se retrouve dans le théorème sur la qualité d'une politique gloutonne fondée sur la fonction de valeur approximée dans (Bertsekas, 1987, p. 236). De même, le principe itératif introduit dans la section 3.1 ressemble au concept d'itération de politique approximée (*approximated policy iteration*, (Bertsekas et Tsitsiklis, 1996)). Finalement, Kakade (2003) introduit l'algorithme d'itération conservative de politique (*Conservative Policy Iteration*), élaboré pour le cas de l'horizon infini et également sans appel à l'expert, qui est très similaire à SEARN.

4. « Apprendre à chercher » : applications au TAL

4.1. *Des applications emblématiques et les espaces de recherche associés*

Les diverses structures linguistiques présentes dans la langue naturelle fournissent un cadre d'application intéressant pour la prédiction structurée et donc pour les méthodes « apprendre à chercher ». Un premier cas de structure très simple est celui de la séquence de symboles, utilisée dans des tâches telles que la reconnaissance de l'écriture manuscrite, la reconnaissance d'entités nommées, l'étiquetage morpho-syntaxique ou le calcul de la prononciation d'une chaîne orthographique. Dans toutes

ces tâches, l'objectif est de mettre en correspondance chaque élément d'une séquence « source » avec un élément d'une séquence cible. Il s'agit d'un cadre très simple d'application pour les méthodes qui nous intéressent, étudié par exemple dans (Daumé *et al.*, 2009 ; Ross *et al.*, 2011 ; Doppa *et al.*, 2014 ; Knyazeva *et al.*, 2015). Pour ces applications, la procédure standard de recherche consiste à effectuer des actions attribuant une étiquette aux différentes positions l'une après l'autre, en utilisant la séquence source et les étiquettes déjà connues pour composer l'état, ce qui permet de mieux informer les actions ultérieures.

La tâche d'analyse en dépendances possède un espace de sorties composé des arbres syntaxiques valides pour une phrase. Dans le cas des analyseurs par transition (Nivre, 2008), la recherche de l'arbre optimal peut être effectuée en réalisant des actions telles que *shift*, *reduce* ou autres[23](Goldberg et Nivre, 2012 ; Chang *et al.*, 2015a ; Chang *et al.*, 2015b). Comme il est typique pour les applications L2S en TAL, l'état est composé de la phrase source ainsi que des informations de toutes les actions déjà effectuées.

La tâche d'extraction d'évènements biomédicaux est similaire : elle vise à fournir pour chaque phrase un graphe dirigé sans cycle, dans lequel les relations entre les mots composant les évènements de différents types sont établies. Vlachos et Craven (2011) l'abordent à l'aide des techniques L2S : lors de l'inférence, plusieurs passes sur la phrase sont effectuées avec comme objectifs successifs de détecter les évènements, leurs arguments et enfin d'établir la structure complète. Chaque sous-tâche pourrait donner lieu à un problème structuré distinct ; la modélisation L2S permet de les intégrer au sein d'une unique recherche, une approche qui est hors de portée des techniques classiques, au vu de la complexité de l'espace de recherche.

La tâche de résolution de coréférences a pour objectif de détecter les formes qui se réfèrent à un même objet « réel » en calculant les regroupements correspondants. Dans (Clark et Manning, 2016), ces regroupements sont construits en réunissant progressivement les groupes de mots en partant des mots isolés et en finissant par un groupe pour chaque référent. Cette approche se prête bien au cadre L2S, avec des actions qui correspondent aux décisions de fusion de groupes de mentions, et un ensemble d'états correspondant aux regroupements partiellement construits.

4.2. *Quelques détails d'implantation*

Les applications d'« apprendre à chercher » au TAL se caractérisent par une certaine liberté dans la construction de l'algorithme, ne correspondant pas toujours au cadre théorique précis. Par exemple, les mises à jour de la politique peuvent être fréquentes plutôt qu'à la fin de chaque époque (Goldberg et Nivre, 2012), le mélange stochastique des politiques dans les méthodes de la famille SEARN peut être construit de manière approximative (Daumé *et al.*, 2009), la contrainte de convexité de la politique

23. L'ensemble des actions possibles dépend de l'architecture de l'analyseur syntaxique.

pour les méthodes de la famille AggraVaTe n'est pas nécessairement remplie (Chang *et al.*, 2015a), etc. Ces applications ont toutefois en commun les grands principes des méthodes L2S, tels que :

– le principe itératif ;

– l'utilisation de la politique expert (*reference roll-in*) ainsi que de la politique apprise (*learned roll-in*) afin d'explorer les états pertinents de l'espace de recherche ;

– l'estimation soigneuse des scores des actions en prenant en compte non seulement la récompense immédiate apportée par l'action, mais aussi son impact global sur la sortie complète (évaluation par l'expert, ou *reference roll-out*) et dans certains cas les imperfections de la politique poursuivant l'action (évaluation par le tuteur, ou *learned roll-out*) ;

– un changement lent de la politique d'une itération à la suivante (ce qui peut être fait par mélange stochastique comme dans SEARN ou par aggrégation des ensembles d'apprentissage comme dans AggreVaTe, ainsi qu'en utilisant les mises à jour en ligne comme proposé dans (Goldberg et Nivre, 2012)).

4.2.1. *Le calcul de l'expert*

Comme expliqué section 2.1.4, les méthodes « apprendre à chercher » utilisent la politique expert pour déterminer l'action optimale à partir d'un état arbitraire de l'espace de recherche (en supposant que les actions suivantes sont elles aussi choisies de manière optimale). En TAL, cette politique peut être plus ou moins simple à calculer selon la structure de l'espace de recherche et de la métrique d'évaluation.

Les tâches d'étiquetage de séquences évaluées en utilisant le score de Hamming disposent d'un expert trivial dans tous les états de l'espace de recherche : c'est l'action proposant l'étiquette de référence pour la position considérée. Dans le cas du F-score, le calcul est également direct (Daumé *et al.*, 2009). Pour certains systèmes d'analyse en dépendances, une méthode de calcul de l'expert en temps constant connue sous le nom « d'oracle dynamique » a été également proposée dans (Goldberg et Nivre, 2012).

Pour d'autres tâches, en revanche, l'expert n'est pas calculable en temps raisonnable. C'est le cas, par exemple, de la résolution de coréférences, où la métrique d'évaluation utilisée est complexe et ne se décompose pas sur les actions isolées (Ma *et al.*, 2014). Cette propriété de non-décomposabilité empêche l'utilisation de programmation dynamique pour calculer efficacement la politique expert. Dans de telles situations, une approximation de la politique expert peut être utilisée, par exemple la politique choisissant l'action apportant l'amélioration maximale immédiate (Clark et Manning, 2016). Une autre approximation possible consiste à utiliser toujours l'étiquette de référence, même si, en combinaison avec les décisions passées, elle n'est pas nécessairement optimale (Venkatraman *et al.*, 2015).

4.2.2. *Le calcul du tuteur*

Le calcul du score du tuteur présente des difficultés différentes de celui de l'expert. D'une part, la politique future étant fixée, l'évaluation concerne donc uniquement

l'action en cours, contrairement au cas de la politique expert qui prend sa décision en prévoyant aussi bien l'action en cours que toutes les autres actions restant à réaliser. D'autre part, il est nécessaire de recalculer ces scores pour chaque nouvelle politique future considérée. Pour les applications au TAL, le score du tuteur est en général calculé en effectuant une simulation reposant sur l'utilisation de la politique dont on souhaite calculer le score (Daumé *et al.*, 2009). Dans la mesure où cette politique peut être stochatique, afin de réduire la variance, cette estimation peut être réalisée à l'aide de plusieurs simulations. Cette reproductibilité est assez particulière au domaine de TAL, elle est due d'une part à la longueur relativement petite des séquences d'actions et d'autre part à l'absence des facteurs aléatoires de l'environnement (ce qui n'est pas le cas par exemple dans la robotique). En revanche, l'utilisation de plusieurs simulations augmente le coût de l'algorithme.

Une approximation souvent employée est l'utilisation de la politique expert à la place de la politique par rapport à laquelle le score doit être calculé (si l'expert est facilement calculable). En pratique cette approximation donne souvent de bons résultats, spécialement si la politique apprise est de qualité proche de l'expert (Daumé *et al.*, 2009 ; Clark et Manning, 2016). Néanmoins, pour certaines tâches le calcul soigneux des scores vis-à-vis de la politique courante se montre important. C'est le cas par exemple pour l'extraction d'évènements biomédicaux (Vlachos et Craven, 2011), ainsi que pour l'analyse en dépendances (Chang *et al.*, 2015b) (pour ce dernier, l'avantage d'utiliser les scores du tuteur, et plus précisément un mélange des scores de l'expert et du tuteur, a été établi pour le cas où la référence est sous-optimale).

4.3. *Motiver l'utilisation de méthodes L2S en TAL*

Pour conclure cette section, notons qu'en TAL plusieurs raisons peuvent motiver l'utilisation de l'inférence approchée par rapport aux méthodes classiques fondées sur la programmation dynamique, ce qui à son tour peut motiver l'utilisation des méthodes « apprendre à chercher »[24].Tout d'abord, la programmation dynamique peut être appliquée uniquement pour les structures relativement simples, comme des séquences et des arbres. Certaines tâches, exemplifiées ici par l'extraction d'évènements biomédicaux, emploient des structures plus complexes, qui sont en dehors du champ d'application de la programmation dynamique (Vlachos et Craven, 2011). On peut de même vouloir utiliser pour l'apprentissage des fonctions de perte non décomposables sur les arcs de l'espace de recherche (comme BLEU (Papineni *et al.*, 2002) en traduction automatique), ce qui reste faisable dans un cadre L2S. Enfin, même si pour certaines tâches, la programmation dynamique permet de réaliser l'apprentissage et l'inférence avec une complexité polynomiale, en pratique cette complexité peut rester prohibitive : c'est le cas de tâches d'étiquetage qui manipulent de très grands ensembles d'étiquettes (Müller *et al.*, 2013 ; Lavergne et Yvon, 2017).

24. En pratique l'utilisation des techniques L2S permet souvent d'améliorer le résultat si la recherche utilisée est gloutonne ou en faisceau, mais ce n'est pas toujours le cas : certains travaux rapportent l'absence d'améliorations perceptibles (Wiseman *et al.*, 2016).

En dehors des problèmes de complexité, l'utilisation de l'inférence approchée peut être motivée par la richesse des informations contenues dans les dépendances longues. Ainsi, il peut être bénéfique de sacrifier l'exactitude de la recherche afin de permettre l'utilisation de dépendances à longue distance, c'est le cas, par exemple, pour l'analyse en dépendances (Zhang et Nivre, 2011 ; Goldberg et Nivre, 2012), l'analyse syntaxique (Collins et Roark, 2004) ou encore des diverses tâches d'étiquetage de séquences (Daumé *et al.*, 2009). Cet aspect peut notamment être renforcé par la modification de la procédure de recherche avec pour objectif de permettre un ordre libre pour la prise de décisions. Ainsi, les décisions complexes (attribution d'une étiquette morphosyntaxique pour un mot ambigu, reconnaissance d'une lettre non clairement écrite) peuvent être retardées et réalisées en utilisant le maximum d'informations sur l'ensemble de la structure (Shen *et al.*, 2007 ; Knyazeva *et al.*, 2015).

Une autre limitation importante imposée par la programmation dynamique est qu'elle restreint fortement la famille de classifieurs pouvant être utilisée, en pratique seuls des classifieurs linéaires semblent être utilisés. Cette limitation peut être en partie contournée en ajoutant des caractéristiques adaptées (manuellement ou bien encore à l'aide de noyaux polynomiaux). Cette « astuce » ne permet toutefois pas d'utiliser des classifieurs plus expressifs tels que les réseaux de neurones récurrents qui restent incompatibles avec les méthodes de programmation dynamique.

5. Conclusion

Dans cet article, nous avons présenté un panorama d'une famille de méthodes permettant de réaliser des tâches d'apprentissage structuré, qui sont au cœur des méthodes du TAL moderne. Contrairement aux méthodes standard d'apprentissage structuré, qui reposent souvent sur des simplifications du modèle de dépendances, les méthodes de la famille L2S s'appuient sur des simplifications de l'algorithme de recherche, supposé effectuer un décodage glouton, ce qui permet d'une part de prendre en compte des dépendances plus complexes, d'autre part de considérer des fonctions de pertes plus riches. Après avoir présenté les bases théoriques qui sous-tendent ces méthodes et illustré leur fonctionnement sur un problème de traduction automatique, nous avons discuté leur utilisation sur diverses tâches du TAL.

Parmi les questions encore ouvertes, mentionnons celles qui s'intéressent à l'utilisation d'autres formes (simples) pour réaliser l'inférence. Il est ainsi connu qu'une manière simple pour améliorer la recherche gloutonne consiste à employer des techniques de recherche en faisceau (*beam search*). La généralisation des méthodes L2S à la recherche en faisceau est difficile, car cette technique considère simultanément plusieurs états de l'espace de recherche : définir dans ce contexte l'équivalent de l'effet de gaspillage de ressources et de l'effet d'avalanche n'est pas immédiat et plusieurs travaux s'inscrivent dans cette lignée pour des tâches d'apprentissage structuré en TAL (Collins et Roark, 2004 ; Huang *et al.*, 2012 ; Aufrant *et al.*, 2017).

Mais l'actualité principale de ces méthodes est liée à l'utilisation de réseaux de neurones récurrents, qui constituent l'état de l'art pour de nombreuses tâches du TAL, comme l'analyse en dépendances (Dyer *et al.*, 2015) et la traduction automatique (Bahdanau *et al.*, 2015). La forte analogie avec le cadre L2S – les décisions sont prises les unes après les autres par « les unités » du réseau récurrent (GRU ou LSTM) jouant un rôle similaire à celui du classifieur local –, et l'impossibilité d'utiliser des méthodes de recherche exacte, font que les concepts et techniques L2S trouvent une claire utilité dans ce cadre. Ainsi, Bengio *et al.* (2015) obtiennent des gains importants pour la tâche de traduction grâce à l'utilisation pendant l'entraînement de la politique apprise ; ces prédictions sont introduites progressivement au cours des itérations, comme dans le cas de SEARN. Ballesteros *et al.* (2016) poursuivent la ligne générale du travail initié par Goldberg et Nivre (2012) en utilisant l'oracle dynamique dans les RNN pour l'analyse en dépendances. Wiseman et Rush (2016) ont adapté une forme plus ancienne d'« apprendre à chercher », LASO (Daumé III et Marcu, 2005), pour l'entraînement des RNN avec une recherche en faisceau et obtenu des succès à la fois en traduction et en analyse en dépendances. Une adaptation d'AggreVaTe a enfin été proposée dans (Sun *et al.*, 2017) avec une modification de l'algorithme d'optimisation sans regret. Ses performances ont été validées sur la tâche d'analyse en dépendances d'équations manuscrites. Dans ces applications, la transposition directe des garanties théoriques des méthodes L2S n'est pas immédiate, comme discuté récemment par Leblond *et al.* (2018) dans leur extension de SEARN aux cas des RNN.

Des questions théoriques et pratiques bien étudiées dans le formalisme L2S se posent de manière proche dans des formalismes apparentés (l'apprentissage par renforcement) comme dans diverses applications utilisant des réseaux récurrents. Nous espérons que la présentation synthétique qui a été donnée ici de ce formalisme plus simple sera utile pour comprendre ces problèmes et leur apporter des réponses idoines.

6. Bibliographie

Aufrant L., Wisniewski G., Yvon F., « Don't Stop Me Now ! Using Global Dynamic Oracles to Correct Training Biases of Transition-Based Dependency Parsers », *Proc. EACL*, Valencia, Spain, p. 318-323, 2017.

Bahdanau D., Cho K., Bengio Y., « Neural machine translation by jointly learning to align and translate », *Proc. ICLR*, 2015.

Bakir G. H., Hofmann T., Schölkopf B., Smola A. J., Taskar B., Vishwanathan S. V. N., *Predicting Structured Data*, MIT Press, 2007.

Ballesteros M., Goldberg Y., Dyer C., Smith N. A., « Training with Exploration Improves a Greedy Stack LSTM Parser », *Proc. EMNLP*, Austin, TX, p. 2005-2010, November, 2016.

Bengio S., Vinyals O., Jaitly N., Shazeer N., « Scheduled Sampling for Sequence Prediction with Recurrent Neural Networks », *Proc. NIPS*, Montreal, Canada, p. 1171-1179, 2015.

Bertsekas D. P., *Dynamic Programming : Deterministic and Stochastic Models*, Prentice-Hall, Inc., 1987.

Bertsekas D. P., Tsitsiklis J. N., *Neuro-Dynamic Programming*, 1st edn, Athena Scientific, 1996.

Chang K., He H., III H. D., Langford J., « Learning to Search for Dependencies », *arXiv preprint arXiv :1503.05615*, 2015a.

Chang K.-W., Krishnamurthy A., Agarwal A., Daumé III H., Langford J., « Learning to search better than your teacher », *Proc. ICML*, Lille, France, p. 2058-2066, 2015b.

Clark K., Manning C. D., « Improving Coreference Resolution by Learning Entity-Level Distributed Representations », *Proc. ACL*, Berlin, Germany, p. 643-653, 2016.

Collins M., Roark B., « Incremental Parsing with the Perceptron Algorithm », *Proc. ACL*, Barcelona, Spain, p. 111-118, July, 2004.

Daumé III H., Langford J., Marcu D., « Search-based Structured Prediction », *Machine Learning*, vol. 75, n° 3, p. 297-325, June, 2009.

Daumé III H., Marcu D., « Learning as Search Optimization : Approximate Large Margin Methods for Structured Prediction », *Proc. ICML*, Bonn, Germany, p. 169-176, 2005.

Doppa J. R., Fern A., Tadepalli P., « HC-Search : A Learning Framework for Search-based Structured Prediction », *JAIR*, vol. 50, p. 369-407, 2014.

Dyer C., Ballesteros M., Ling W., Matthews A., Smith N. A., « Transition-Based Dependency Parsing with Stack LSTM », *Proc. ACL-IJCNLP*, Beijing, China, p. 334-343, 2015.

Elkan C., « The Foundations of Cost-sensitive Learning », *Proceedings of IJCAI 2001*, San Francisco, CA, USA, p. 973-978, 2001.

Geist M., Contrôle optimal et apprentissage automatique, applications aux interactions homme-machine, Habilitation à diriger des recherches, Université de Lille 1, Février, 2016.

Goldberg Y., Nivre J., « A Dynamic Oracle for Arc-Eager Dependency Parsing », *Proceedings of COLING 2012*, Mumbai, India, p. 959-976, December, 2012.

Huang L., Fayong S., Guo Y., « Structured Perceptron with Inexact Search », *Proc. ACL-HLT*, Montréal, Canada, p. 142-151, June, 2012.

Kakade S., On the Sample Complexity of Reinforcement Learning, PhD thesis, Gatsby Computational Neuroscience Unit, University College London, 2003.

Knyazeva E., Wisniewski G., Yvon F., « Apprentissage par imitation pour l'étiquetage de séquences : vers une formalisation des méthodes d'étiquetage "easy-first" », *Proc. TALN*, Caen, France, p. (12), 2015.

Lavergne T., Yvon F., « Learning the Structure of Variable-Order CRFs : a finite-state perspective », *Proc. EMNLP*, Copenhagen, Denmark, p. 433-439, 2017.

Leblond R., Alayrac J.-B., Osokin A., Lacoste-Julien S., « SEARNN : Training RNNs with global-local losses », *Proc. ICLR*, 2018.

Liang P., Bouchard-Côté A., Klein D., Taskar B., « An End-to-end Discriminative Approach to Machine Translation », *Proc. COLING-ACL*, Sydney, Australia, p. 761-768, 2006.

Lê M., Fokkens A., « Tackling Error Propagation through Reinforcement Learning : A Case of Greedy Dependency Parsing », *Proc. EACL*, Valencia, Spain, p. 677-687, 2017.

Ma C., Doppa J. R., Orr J. W., Mannem P., Fern X. Z., Dietterich T. G., Tadepalli P., « Prune-and-Score : Learning for Greedy Coreference Resolution », *Proc. EMNLP*, Doha, Qatar, p. 2115-2126, 2014.

Müller T., Schmid H., Schütze H., « Efficient Higher-Order CRFs for Morphological Tagging », *Proc. EMNLP*, Seattle, Washington, USA, p. 322-332, 2013.

Nivre J., « Algorithms for Deterministic Incremental Dependency Parsing », *Computational Linguistics*, vol. 34, n° 4, p. 513-553, 2008.

Papineni K., Roukos S., Ward T., Zhu W.-J., « BLEU : A Method for Automatic Evaluation of Machine Translation », *Proc. ACL*, Philadelphia, Pennsylvania, p. 311-318, 2002.

Puterman M. L., *Markov Decision Processes : Discrete Stochastic Dynamic Programming*, 1st edn, John Wiley & Sons, Inc., 1994.

Ranzato M., Chopra S., Auli M., Zaremba W., « Sequence Level Training with Recurrent Neural Networks », *Proc. ICLR*, 2016.

Ross S., Bagnell J. A., « Reinforcement and Imitation Learning via Interactive No-Regret Learning », *arXiv preprint arXiv :1406.5979v1*, 2014.

Ross S., Gordon G., Bagnell D., « A Reduction of Imitation Learning and Structured Prediction to No-Regret Online Learning », *Proc. AIStats*, vol. 15, Fort Lauderdale, FL, p. 627-635, 2011.

Shen L., Satta G., Joshi A., « Guided Learning for Bidirectional Sequence Classification », *Proc. ACL*, Prague, Czech Republic, p. 760-767, 2007.

Smith N. A., *Linguistic Structure Prediction*, 1st edn, Morgan & Claypool Publishers, 2011.

Sokolov A., Wisniewski G., Yvon F., « Lattice BLEU Oracles in Machine Translation », *ACM Transactions on Speech and Language Processing*, vol. 10, n° 4, p. 18 :1-18 :29, 2013.

Sun W., Venkatraman A., Gordon G. J., Boots B., Bagnell J. A., « Deeply AggreVaTeD : Differentiable Imitation Learning for Sequential Prediction », *in* D. Precup, Y. W. Teh (eds), *Proc. ICML*, vol. 70, Sydney, Australia, p. 3309-3318, 2017.

Sutton R. S., Barto A. G., *Introduction to Reinforcement Learning*, 1st edn, MIT Press, 1998.

Tellier I., Tommasi M., « Champs Markoviens Conditionnels pour l'extraction d'information », *in* E. Gaussier, F. Yvon (eds), *Modèles probabilistes pour l'accès à l'information textuelle*, Hermès, p. 223-267, 2011.

Venkatraman A., Hebert M., Bagnell J. A., « Improving Multi-step Prediction of Learned Time Series Models », *Proc. AAAI*, Austin, TX, p. 3024-3030, 2015.

Viterbi A. J., « Error Bounds for Convolutional Codes and an Asymptotically Optimum Decoding Algorithm », *IEEE Trans. on Information Theory*, vol. 13, n° 2, p. 260-269, 1967.

Vlachos A., Craven M., « Search-based Structured Prediction Applied to Biomedical Event Extraction », *Proc. CoNLL*, Portland, OR, p. 49-57, 2011.

Wiseman S., Rush A. M., « Sequence-to-Sequence Learning as Beam-Search Optimization », *Proc. EMNLP*, Austin, TX, p. 1296-1306, November, 2016.

Wiseman S., Rush A. M., Shieber S. M., « Learning Global Features for Coreference Resolution », *Proc. NAACL-HLT*, San Diego, CA, p. 994-1004, 2016.

Zhang Y., Nivre J., « Transition-based Dependency Parsing with Rich Non-local Features », *Pro. ACL-HLT*, Portland, Oregon, p. 188-193, 2011.