Algorithme de recherche d'un rang de prédiction. Application à l'évaluation de modèles de langage

P. Alain, O. Boëffard

IRISA / Université de Rennes 1 - ENSSAT
6 rue de Kerampont, B.P. 80518, F-22305 Lannion Cedex
{pierre.alain,olivier.boeffard}@irisa.fr
http ://www.irisa.fr/cordial/

ABSTRACT

Within a predictive framework for language model evaluation, Shannon uses a rank distribution in order to bound the entropy of printed English. Taking into account of higher dimensions (prediction of symbols in a raw) and predicting a k-word sequence given a N-word vocabulary is a NP-hard computational task. To achieve this goal, we propose some acceptable and effective search heuristics for an A^* algorithm.

1. INTRODUCTION

De nombreux systèmes de traitement de l'information font appel aux modèles de langage. Les domaines d'application sont variés, il s'agit notamment du traitement de la parole, de la traduction automatique, ou de la recherche d'information. L'évaluation des modèles de langage repose sur deux postures méthodologiques relativement tranchées. Une première adopte pour principe qu'il est nécessaire d'évaluer un modèle dans son contexte d'application. Rosenfeld [6] considère à cet égard qu'il faut parler d'un gain de 10% sur la perplexité pour obtenir des résultats significatifs notamment en reconnaissance de la parole. Une seconde approche définit un cadre méthodologique qui ne nécessite pas l'application visée. Le critère le communément admis est celui de l'entropie-croisée, exprimé sous la forme d'une mesure de perplexité. Le principal inconvénient du critère de perplexité est qu'il fait l'hypothèse d'un modèle probabiliste. Comme corollaire à cette nécessité, la mesure de perplexité est directement reliée au nombre de paramètres du modèles. Notamment, plus le nombre de degré de liberté d'un modèle augmente meilleurs sont les scores de perplexité. Avec ce seul critère, il est donc impossible de juger à la fois de la précision d'un modèle et de son efficacité (au sens d'une longueur de description minimale par exemple), sauf à mettre en place un scénario d'évaluation plus complexe de validation croisée.

Les travaux que nous proposons se situent sur le deuxième axe méthodologique. Nous pensons en effet qu'il est plus pertinent d'évaluer un modèle pour ses qualités propres qu'au travers d'une application. Une évaluation qui dépend de la tâche ne permet pas, en toute rigueur, de généraliser des résultats à des domaines connexes. Dans [3], nous avions fait le choix de juger de la performance d'un modèle de langage selon ses capacités de prédiction. Ce scénario méthodologique reprend les idées de Shannon [7], nous les avons étendues à la prédiction conjointe de séquences de mots de manière à couvrir les différents types de modèle de langage proposés par la littérature (notamment ceux qui prédisent plusieurs mots en un seule étape de traitement). Bimbot [2] a déjà fait allusion au cadre de Shannon mais reste sur une estimation d'une mesure de perplexité.

Cet article propose une solution algorithmique à l'évaluation du rang de prédiction d'une séquence de mots étant connu un historique. Le prédicteur, ou modèle de langage, est ici très général, il peut s'agir de modèles probabilistes ou non. Le problème est de nature combinatoire. La difficulté provient du fait que la décision de prédiction sur un mot dépend des prédictions déjà réalisées. Nous n'avons pas la place de présenter à la fois les détails algorithmiques et une évaluation expérimentale. Nous présentons ici les démonstrations d'admissibilité des fonctions d'élagage appliquées à l'algorithme de recherche des meilleurs chemins construit sur une base A^*.

Au cours du paragraphe 2, nous présentons la problématique du calcul des rangs de prédiction ainsi qu'une formalisation par des graphes multivalués. Dans la section 3 nous présentons des algorithmes de parcours pour un graphe mulivalué. La section 4 présente les fonctions d'élagage qui ont été mises en place. Nous conclurons dans la section 5.

2. MODÉLISATION DU PROBLÈME

Soit une séquence de l mots de test, la détermination du rang de prédiction de cette séquence par un modèle de langage passe par une modélisation sous forme de graphe. On reprend les notations proposé par [1]. On note G un graphe, s le nœud source qui relie la fin des mots de l'historique au début de la fenêtre de prédiction. On note $c(i, j)$ le coût d'un arc du nœud i au nœud j, il s'agit donc du coût d'émission d'un mot, et $c(P, i)$ le coût entre s et i sur le chemin P.

La figure 1, partie a) présente le graphe qui correspond à la recherche des chemins de prédiction des trois mots $[W_i\ W_{i+1}\ W_{i+2}]$. Cette prédiction peut s'effectuer mot-à-mot en utilisant les arcs de coût $b(W_i)$, $b(W_{i+1})$ ou encore $b(W_{i+2})$. Ce type de cheminement dans le graphe est isomorphe à un calcul de perplexité. On peut également choisir de prédire deux mots en une seule fois avec les arcs de coût $b(W_i, W_{i+1})$ ou $b(W_{i+1}, W_{i+2})$. Enfin, on peut directement prédire les trois mots en une seule fois avec l'arc de coût $b(W_i, W_{i+1}, W_{i+2})$.

Ce graphe est *multivalué*. Chaque arc représente en réalité un fuseau d'arcs qui correspond à l'ensemble des historiques possibles que le modèle de langage peut utiliser pour effectuer sa prédiction, figure 1 partie b). Les arcs

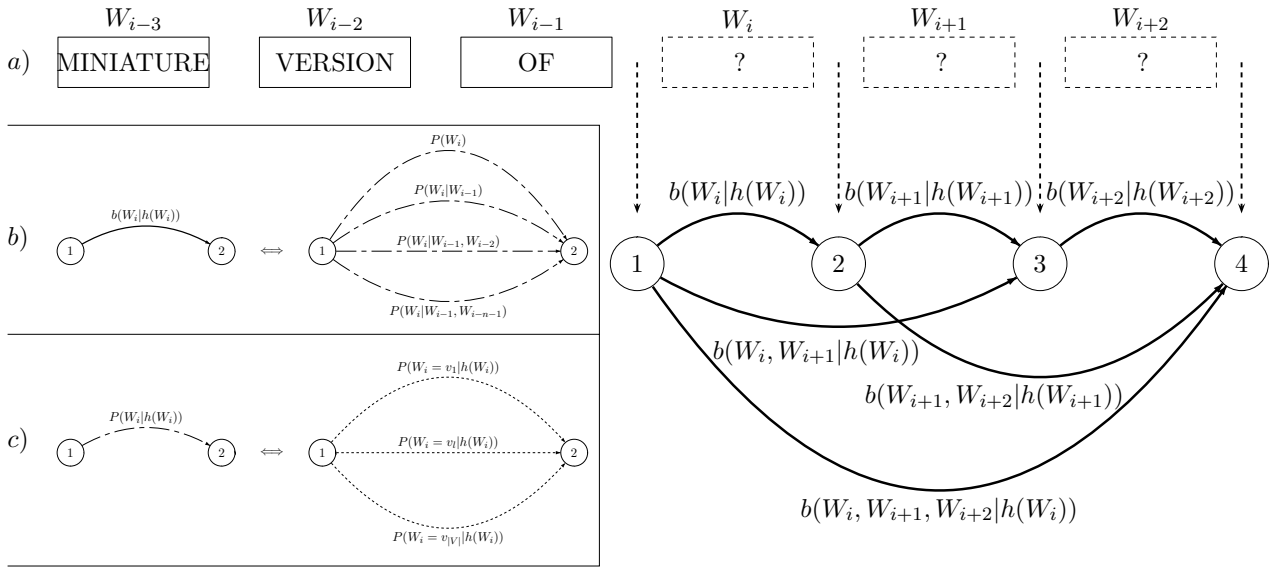

FIG. 1: Le graphe de recherche de chemins pour la prédiction de trois mots à partir d'un contexte connu $[MINIATURE\ VERSION\ OF]$ (a)). La partie $b)$ présente le fuseau en fonction de l'historique, la partie $c)$ le fuseau en fonction du mot prédit.

qui composent ces fuseaux sont eux-mêmes des fuseaux d'arcs. Chaque arc, au dernier niveau, correspond à l'ensemble des mots connus du modèle ; c'est-à-dire à l'ensemble des mots du vocabulaire, plus précisément à l'ensemble des mots prédictibles, figure 1 partie c).

Le graphe de prédiction est un graphe multivalué *dirigé*, la prédiction se fait toujours sur un ou plusieurs mots du futur, les arcs sont alors tous dirigés dans le même sens. Le graphe est également *non absorbant*, car le coût d'un arc est toujours de même signe. En empruntant un nouvel arc, on ne baisse jamais le coût total du chemin.

La propriété de multivaluation entraîne un *non déterminisme*. Il existe plusieurs arcs qui permettent de prédire le mot W_i. Ce graphe peut être transformé en un graphe déterministe par démultiplication des nœuds en fonction de l'historique et du mot prédit. Le graphe déterministe ainsi construit possède un nombre exponentiel de nœuds.

La détermination du rang de prédiction R d'une séquence de mots de test correspond à la recherche de tous les meilleurs chemins de prédiction du modèle de langage jusqu'au chemin qui correspond à l'émission des mots de test. Le rang de prédiction R est donné par le rang de classement du chemin des mots de test. Le rang 1 correspond à l'émission la plus probable du modèle de langage (qui n'est pas forcément la séquence des mots de test).

Nous venons de caractériser par un graphe un modèle de description qui tient compte des différentes alternatives proposées par un modèles de langage. Il nous reste à définir une procédure d'énumération de ces alternatives de prédiction pour ensuite les classer entre elles en fonction de leurs rangs.

3. PARCOURS DU GRAPHE

On s'intéresse tout d'abord aux algorithmes de recherche de la meilleure séquence de prédiction. La recherche de cette meilleure séquence correspond à une recherche du meilleur chemin dans le graphe décrit précédemment. On peut montrer que des algorithmes de décodage classiques, comme l'algorithme de Dijkstra, ne peuvent cependant pas s'appliquer à un graphe multivalué.

Nous avons dans un premier temps adapté un algorithme A^* pour la recherche d'un meilleur chemin dans un graphe multivalué, section 3.1. Cette première base algorithmique a ensuite été complétée pour le traitement des R meilleurs chemins, section 3.2. Cette section montre comment cet algorithme ne peut éviter l'explosion combinatoire lors de la recherche car nous ne connaissons pas la valeur de coût du chemin au rang R. L'alternative que nous proposons à cette contrainte combinatoire sera développée section 4 et consiste en une fonction efficace d'élagage des chemins de rang supérieur à celui de la séquence des mots de test.

3.1. Algorithme A^*, principe

L'algorithme A^*, dans le cas général, conserve au niveau de chaque nœud un ensemble d'hypothèses. Cet ensemble est une mémoire des meilleurs chemins qui permettent d'atteindre un nœud n depuis le nœud s. Deux listes globales permettent de conserver ces informations : une liste $OPEN$ de nœuds à explorer et une liste $CLOSE$ de noeuds déjà explorés.

A^* se base sur une heuristique qui estime le coût du chemin optimal d'un nœud n au nœud final. Pour donner le résultat optimal, cette heuristique ne doit en aucun cas surestimer le coût global jusqu'au nœud final. L'algorithme est alors dit admissible. Si $h^*(n)$ est le coût réel du chemin du mot n jusqu'à la fin de la fenêtre de prédiction, et $h(n)$ le coût proposé par une heuristique admissible : $h(n) \leq h^*(n)$) [4].

Pour cette implantation à heuristique nulle, l'algorithme A^* est équivalent à l'algorithme de Dijkstra, mais l'utilisation de la liste $OPEN$ permet de contourner le problème du non déterminisme du graphe multivalué. La complexité en terme de nombre de nœuds s'est transformée en une complexité en terme d'espace pour le tas qui permet d'im-

planter la liste $OPEN$. Un gain en complexité spatiale peut apparaître dans la mesure ou l'algorithme A^* n'a pas besoin d'explorer tous les chemins possibles du graphe avant de trouver la meilleure ou les R meilleures solutions. Cette propriété sera systématisée, section 4 par la mise en place d'une fonction d'élagage des mauvais chemins.

3.2. Recherche des R meilleurs chemins

L'algorithme A^* garde sur chaque nœud une liste ordonnée des meilleurs hypothèses qui permettent d'atteindre un nœud n depuis le nœud source. Il est donc très facile de passer d'une version algorithmique de recherche d'un meilleur chemin à une recherche des R meilleurs [5].

Le déroulement de l'algorithme est le suivant : on initialise un tas avec le nœud s de coût d'accès nul, puis on itère sur le procédé suivant : on enlève du tas le chemin de coût le plus faible (en fait le nœud final du chemin dont le coût d'accès est le plus faible), puis on met dans le tas tous les candidats, mots possibles selon le modèle, qui peuvent apparaître à la suite du nœud extrait. Une heuristique permet de tenir compte du coût reel du chemin déjà parcouru, $c(P, i)$, du coût de au nœud voisin ($c(i, j)$), et d'une estimation du coût du chemin jusqu'au nœud de destination ($h(j)$) . Comme déjà discuté, on prendra la valeur nulle comme estimation du coût futur. On obtient ainsi une implantation capable de fonctionner avec un graphe multivalué.

À l'exécution, ajouter tous les candidats possibles fait croître très rapidement la taille du tas. Pour cette raison, un élagage doit être entrepris. Le principe consiste à vérifier que le coût d'un chemin passant par le nœud courant jusqu'au nœud voisin reste inférieur à une certaine valeur de seuil. Toute la difficulté de cette méthodologie est que lors de la recherche des meilleurs chemins, on ne connaît pas encore R. Si nous ne recherchons que les k meilleurs chemins, nous devons nous assurer que le rang de la fenêtre de prediction R est bien tel que $k \geq R$.

4. A^* ET HEURISTIQUE D'ÉLAGAGE

Nous avons constaté qu'un graphe multivalué peut conduire à plusieurs chemins possibles pour une séquence de mots de test. Pour une fenêtre de prédiction, le rang des mots de la séquence de test, $W_i, \cdots, W_{i+l}$, est celui du chemin qui émet cette séquence avec le coût le plus faible. Nous présentons tout d'abord, section 4.1, l'algorithme A^* mis en œuvre. Ensuite deux heuristiques d'élagage sont proposées pour limiter l'explosion combinatoire de la recherche de R meilleurs chemins, section 4.2.

4.1. Présentation de l'algorithme

Le principe de l'élagage consiste à ne pas inclure dans la liste $OPEN$ des nœuds qui appartiennent à des chemins dont le rang sera supérieur au coût le plus faible des chemins qui correspondent aux mots de test de la fenêtre de prediction. C'est-à-dire qu'aucun chemin en cours de construction par A^* ne doit être éliminé s'il peut obtenir un rang meilleur que celui de la fenêtre de prediction. L'algorithme utilisé est le suivant :

N_{sol} représente le nombre de solutions déjà trouvées, Sol est l'ensemble de ces solutions. $ELAGAGE$ correspond

Algorithme 1

```
1   N_sol ⇐ 0
2   Sol ⇐ ∅
3   OPEN ⇐ {s}
4   while (¬(heap = ∅ ∨ is_sol(last(Sol)) ∨
5                          N_sol = N_max)) do begin
6    path ⇐ min(OPEN)
7    if (path_length = l) begin
8     Sol ⇐ Sol ∪ {path}
9     N_sol ⇐ N_sol + 1
10   end else begin
11    forall (node ∈ next(path)) do begin
12     P ⇐ path ⊕ node
13     if (c(P, node) < ELAGAGE) begin
14      OPEN ⇐ OPEN ∪ {newPath}
15     end
16    end
17   end
18  end

19  if (is_sol(last(Sol))) begin
20   Le rang de la séquence est N_sol
21  end else begin
22   La séquence n'a pas été trouvée
23  end
```

à l'heuristique d'élagage, voir section 4.2 pour les détails d'implantation. $heap \Leftarrow \{s\}$ indique que le tas est initialisé avec le nœud source, noeud fictif qui correspond à la fin du contexte connu (nœud 1 sur la figure 1). Parmi les opérateurs utilisés, $last(Sol)$ donne la dernière solution ajoutée dans Sol si Sol est non vide, $null$ sinon. $is\_sol(.)$ renvoie vrai si le paramètre en entrée correspond à la sequence de prédiction ($is\_sol(null) = false$). $min(.)$ retourne et supprime du tas le chemin de coût minimal. $next(.)$ est une fonction qui à partir d'un nœud donne la liste de ses voisins : cette fonction permet de traverser le graphe. $\oplus$ permet d'ajouter un nœud à un chemin.

La paragraphe suivant établit des propositions pour donner une valeur au seuil d'élagage $ELAGAGE$.

4.2. Fonctions d'élagage

Avant de décrire et comparer les deux fonctions d'élagage proposées, la proposition suivante définit un domaine de recherche de ces constantes.

Proposition 1 *Soit une fonction d'élagage constante c supérieure au coût du chemin des mots de test dans la fenêtre de prédiction. Alors les seuls chemins élagués sont de rangs strictement supérieurs à celui de la fenêtre de prédiction.*

Preuve : Soit P un chemin élagué par la fonction d'élagage c, le coût de P est donc strictement supérieur à c. D'après le choix de c, le chemin P a un coût strictement supérieur à celui de la fenêtre de prédiction. Par conséquent, le rang de P est strictement supérieur à celui de la fenêtre de prédiction. □

On peut donc choisir comme premier seuil d'élagage le coût du chemin qui correspond à la prédiction des mots de test. On se retrouve à poser comme heuristique le coût d'une séquence de mots évalué mot à mot.

En partant du premier nœud, jonction entre l'historique connu et le début de la fenêtre de prédiction, on demande

au modèle de langage d'évaluer le coût de prédiction du premier mot de la fenêtre. On ajoute ce premier mot à l'historique dont se sert le modèle pour évaluer le coût de prédiction du mot suivant. On continue ainsi jusqu'à la fin de la fenêtre. On obtient alors une évaluation du coût c_1 de prédiction de cette fenêtre de test. On définit ainsi la fonction d'élagage constante c_1. On rappelle que le coût de la fenêtre de prédiction est défini comme étant le minimum des coûts des chemins menant à l'émission des mots de test de la fenêtre de prédiction. Par conséquent, la fonction d'élagage c_1 vérifie la proposition 1.

Une seconde proposition consiste à exécuter l'algorithme de Dijkstra de manière à ce que le coût de prédiction intègre toutes les alternatives possibles du modèle de langage, notamment la prédiction conjointe de plusieurs mots. L'algorithme de Dijkstra se révèle nécessaire car il faut réaliser un décodage de tous les chemins qui peuvent aboutir à la prédiction des mots de test et conserver celui de coût minimal. On note c_2 ce coût minimal donné par l'algorithme de Dijkstra. De même que pour c_1, le coût c_2 est supérieur au coût de la fenêtre de prédiction, par la définition de celle-ci. Par conséquent, la fonction d'élagage c_2 satisfait également la proposition 1.

Proposition 2 *La fonction d'élagage c_2 est plus efficace que la fonction c_1.*

Preuve : Le chemin emprunté lors du calcul de c_1 fait parti de l'ensemble des chemins vus lors du décodage de la fenêtre pendant l'exécution de l'algorithme de Dijkstra pour le calcul de c_2. On trouve donc $c_2 \leq c_1$ et la fonction d'élagage c_2 est plus sélective que celle définie par c_1. □

Corollaire 1 *Lorsque l'algorithme 1 donne le rang de la fenêtre de prédiction, ce rang est le bon.*

Preuve : D'après la proposition 1, tous les chemins élagués ont un coût plus élevé que le meilleur chemin qui correspond aux mots de test ; un arrêt par $is\_empty(heap)$ est donc impossible, et si on définit N_{max} de façon à ne pas atteindre la borne, l'algorithme sort avec $is\_Sol(last(Sol))$ à vrai. Cela signifie qu'alors tous les chemins d'un coût inférieur a celui des mots de test ont été prédits, et le rang obtenu est celui de la fenêtre de prédiction. □

La figure 2 montre sur un exemple l'évolution du coût de prédiction d'une fenêtre de trois mots, dans le cadre du calcul de c_1 (courbe du haut), et dans le cadre du calcul de c_2 (courbe du bas). Sur cet exemple, on note que $c_2 < c_1$. Cette différence est due à la prédiction de deux mots conjoints $[HIGH\ YIELD]$. La coupe réalisée par l'élagage est marquée des hachures ; cela signifie qu'un chemin dont le coût passe au dessus de la zone délimitée par c_2 n'est pas à intégrer dans la liste $OPEN$ de A^*. On utilisera donc la fonction $ELAGAGE = c_2$ dans l'algorithme 1 lors de la recherche R meilleurs chemins.

5. CONCLUSION

Dans cet article, nous avons présenté deux fonctions d'élagage pour l'algorithme A^* permettant de rechercher le rang auquel est prédit un chemin particulier d'un multigraphe. Ces fonctions se basent sur la connaissance du coût du chemin dont on cherche le rang pour éliminer les chemins de rang supérieur. L'élagage réalisé permet de limiter la combinatoire de l'algorithme. Cet algorithme A^* a été utilisé dans le cadre d'une évaluation prédictive de modèles de langage. Des distributions de rangs peuvent être comparés sur la moyenne obtenue, plus le rang moyen est faible, plus le modèle est bon prédicteur.

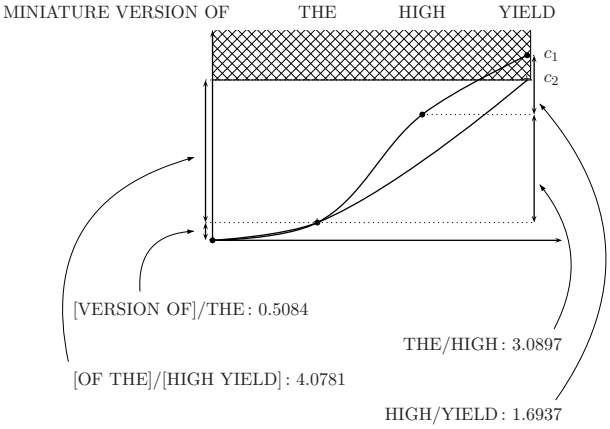

FIG. 2: Calcul des heuristiques c_1 et c_2 pour la prédiction de la fenêtre $[THE\ HIGH\ YIELD]$. c_2 est un coût plus faible que c_1, car la méthode d'estimation de c_1 ne voit pas le chemin qui prédit $[HIGH\ YIELD]$ en une seule fois, alors que l'algorithme de Dijkstra le voit lors du calcul de c_2. Un élagage peut être réalisé à partir du coût c_2 (hachures).

RÉFÉRENCES

[1] A. Bagchi and A. Manhanti. Search algorithms under different kind of heuristics : A comparative study. *Journal of the Associations for Computing Machinery*, 30 :1–21, 1983.

[2] F. Bimbot, M. El-Beze, S. Igounet, M. Jardino, K. Smaili, and I. Zitouni. An alternative scheme for perplexity estimation and its assessment for the evaluation of language models. *Computer Speech and Language*, 15(1) :1–13(13), 2001.

[3] O. Boëffard and P. Alain. Comparing rank-based statistics and standard perplexity to evaluate statistical language models. In *Proceedings of the International Conference on Speech and Computer*, pages 111–114, 2005.

[4] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of a^*. *Journal of the Associations for Computing Machinery*, 32(3) :505–536, 1985.

[5] D. Eppstein. Finding the k shortest paths. *SIAM J. Computing*, 28(2) :652–673, 1998.

[6] R. Rosenfeld. Two decades of statistical language modeling : where do we go from here ? *Proceedings of the IEEE*, 88(8) :1270–1278, 2000.

[7] C. Shannon. Prediction and entropy of printed english. *Bell System Technical Journal*, 30 :50–64, 1951.