Etude et conception d'un correcteur orthographique pour la langue haoussa

Lawaly Salifou et Harouna Naroua

Département de Mathématiques et Informatique, Faculté des Sciences et Techniques
Université Abdou Moumouni, BP 10662 – Niamey, NIGER
salifoumma@yahoo.fr, hnaroua@yahoo.com

Résumé. Dans cet article, un correcteur d'orthographe a été conçu, développé et testé pour la langue haoussa qui est la deuxième langue la plus parlée en Afrique et ne disposant encore pas d'outils de traitement automatique. La présente étude est une contribution pour le traitement automatique de la langue haoussa. Nous avons mis en œuvre les techniques et méthodes prouvées pour d'autres langues afin de concevoir un correcteur orthographique pour le haoussa. Le correcteur conçu au bout de ce travail exploite pour l'essentiel le dictionnaire de Mijinguini et les caractéristiques de l'alphabet haoussa. Après un état des lieux sur la correction orthographique et l'informatisation du haoussa, nous avons opté pour les structures de données trie et table de hachage pour implanter le dictionnaire. La distance d'édition et les spécificités de l'alphabet haoussa ont été mises à profit pour traiter et corriger les erreurs d'orthographe. L'implémentation du correcteur orthographique a été faite sur un éditeur spécial développé à cet effet (LyTextEditor) mais aussi comme une extension (add-on) pour OpenOffice.org. Une comparaison a été faite sur les performances des deux structures de données utilisées.

Abstract. In this paper, we have designed, implemented and tested a spell corrector for the Hausa language which is the second most spoken language in Africa and do not yet have processing tools. This study is a contribution to the automatic processing of the Hausa language. We used existing techniques for other languages and adapted them to the special case of the Hausa language. The corrector designed operates essentially on Mijinguini's dictionary and characteristics of the Hausa alphabet. After a careful study of the existing spell checking and correcting techniques and the state of art in the computerization o f the Hausa language, we opted for the data structures trie and hash table to represent the dictionary. We used the edit distance and the specificities of the Hausa alphabet to detect and correct spelling errors. The implementation of the spell corrector has been made on a special editor developed for that purpose (LyTexEditor) but also as an extension (add-on) for OpenOffice.org. A comparison was made on the performance of the two data structures used.

Mots-clés: Traitement automatique des langues, informatisation du haoussa, langues africaines, correcteur orthographique.

Keywords: Natural Language Processing, computerization of Hausa, African languages, spell checker, spell corrector .

1 Introduction

Le traitement automatique des langues naturelles (TALN) a plusieurs applications industrielles dont, entre autres, la vérification et la correction de l'orthographe et de la grammaire, l'indexation de texte et l'extraction d'informations à partir d'Internet, la reconnaissance vocale, la synthèse vocale, le contrôle vocal des robots domestiques, les systèmes de réponse automatique et la traduction automatique (Kukich, 1992) et (Pierre, 2006). Parmi ces applications, la correction orthographique est de loin la plus répandue. En effet, elle est intégrée à des outils informatiques utilisés chaque jour par des millions de personnes à travers le monde. Les programmes informatiques concernant l'orthographe sont de deux sortes : les vérificateurs d'orthographe et les correcteurs orthographiques. Un vérificateur d'orthographe détecte, dans un texte donné en entrée, les mots qui sont incorrects. Un correcteur orthographique détecte en même temps les erreurs d'orthographes et cherche le mot correct le plus probable (Peterson, 1980). La correction peut être automatique, dans le cas d'un synthétiseur vocal par exemple, ou interactif permettant à l'utilisateur de choisir le mot voulu parmi plusieurs suggestions (Kukich, 1992). Cette deuxième approche est celle de la plupart des logiciels de traitement de texte. Des tels programmes sont généralement conçus pour fonctionner pour une langue donnée. La correction orthographique est, de

nos jours, quasi-présente dans toutes les applications informatiques où du texte est appelé à être entré par l'utilisateur. Celui-ci est généralement avisé d'une saisie incorrecte par un soulignement en rouge du mot erroné. Comme exemples de telles applications, nous pouvons citer : les logiciels de traitement de texte, les clients de messagerie, les éditeurs de code source et les environnements de programmation, les moteurs de recherche sur Internet. Les causes d'erreurs sont de plusieurs ordres et on rencontre plus d'une façon de les classifier (Suzan, 2002). Les plus importantes causes sont l'ignorance de l'auteur, les erreurs typographiques et les erreurs de transmission et de stockage (Peterson, 1980). Un correcteur orthographique accomplit deux fonctions essentielles, l'une après l'autre : la détection d'abord et ensuite la correction d'erreurs d'orthographe. Les méthodes de détection et de correction fonctionnent selon trois approches (Kukich, 1992):

- La détection d'erreurs consistant en des mots orthographiquement étrangers à la langue par exemple 'grafe' écrit à la place de 'girafe'.
- La correction de mot isolé qui consiste à corriger le mot précédemment détecté en le considérant seul sans tenir compte des mots qui l'entourent.
- La détection et la correction contextuelles d'erreurs où chaque mot est considéré en tenant compte du contexte. Ce qui permet de corriger les erreurs orthographiques même quand elle consiste en des mots présents dans la langue mais qui sont mal placés. C'est le cas par exemple du mot 'dessert' saisi à la place de 'désert'.

2 Techniques et algorithmes de détection et de correction d'erreurs

La recherche de solutions au problème de correction orthographique de texte est restée, depuis longtemps, un défi. Plusieurs chercheurs se sont penchés sur le problème et, grâce à leurs efforts, diverses techniques et de nombreux algorithmes ont vu le jour. La détection d'erreurs consiste à trouver les mots orthographiquement incorrects dans un texte. Un mot considéré comme erroné est alors marqué par l'application chargée de vérifier l'orthographe. Si le mot est vraiment erroné – parce que ce n'est pas toujours le cas – on dit qu'une erreur est détectée. Les recherches dans ce domaine ont été effectuées par de nombreux auteurs comme (Enguehard et al., 2011). Les principales techniques utilisées pour l'identification de mots erronées dans un texte sont soit basées sur l'analyse des n-grammes, soit sur la recherche dans un dictionnaire (Kukich, 1992). Un algorithme pour la détection à base d'un dictionnaire est donné par (Peterson, 1980).

La table de hachage est l'une des structures de données la plus utilisée pour réduire le temps de réponse lors de la recherche dans un dictionnaire (Kukich, 1992). L'idée de la table de hachage fut introduite pour la première fois en 1953 (Knuth, 1973). Elle a l'avantage de permettre, grâce au code de hachage, un accès sélectif au mot recherché. Ce qui réduit considérablement le temps de réponse. Mais l'inconvénient majeur est de trouver une fonction de hachage qui admette très peu de collisions et qui donne des indices régulièrement répartis dans l'intervalle considéré.

Les arbres binaires de recherche sont surtout utiles pour vérifier si un mot donné fait partie d'un ensemble plus large de mots qui est ici le dictionnaire. Il existe plusieurs variantes d'arbres binaires de recherche qui ont été utilisés aux fins d'accélérer la recherche dans un dictionnaire dans le cadre de la vérification orthographique.

Les automates finis ont également été utilisés dans certains algorithmes de recherche dans un dictionnaire ou dans un texte. L'un des algorithmes célèbres dans ce domaine est celui de (Aho, Corasick, 1975). L'algorithme consiste à avancer dans une structure de données abstraite appelée dictionnaire qui contient le ou les mots recherchés en lisant les lettres du texte une par une. La structure de données est implantée de manière efficace, ce qui garantit que chaque lettre du texte n'est lue qu'une seule fois. Généralement le dictionnaire est implanté à l'aide d'un trie ou arbre digital auquel on rajoute des liens suffixes. Un trie peut être vu comme la représentation de la fonction de transitions d'un automate fini déterministe. Une fois le dictionnaire implanté, l'algorithme a une complexité linéaire en la taille du texte et des chaînes recherchées.

Bien que la technique des n-grammes calculés à partir d'un dictionnaire soit bonne, elle offre moins de précision que les techniques utilisant toutes les informations du dictionnaire. Mais ces dernières s'avèrent gourmandes en temps lorsque la structure de données implémentant le dictionnaire est mal choisie. Une étude comparative a prouvé que la table de hachage offre des meilleures performances que l'AVL tree, le Red-Black tree et la Skip list (Mark, 2009). La comparaison de cinq structures de données a été effectuée dans le cadre du dictionnaire Punjabi (Lehal, Singh, 2000). Il s'agit de l'arbre binaire de recherche, le trie, le 'ternary search tree', le 'multi-way tree' et le 'reduced memory method tree'. Il en résulte que l'arbre binaire de recherche (ABR) est la structure de donnée la plus convenable en termes de

mémoire utilisée et de temps. Mais l'ABR est limité lorsqu'il s'agit de suggérer une liste de candidats pour la correction ou de trouver tous les mots différents d'une ou de deux lettres. Cette limitation peut être levée par l'utilisation d'un trie qui offre pratiquement la même complexité en temps que l'ABR. Les deux structures de données les mieux indiquées pour implémenter un dictionnaire seraient la table de hachage et le trie.

La correction d'erreurs fait référence au fait de doter les vérificateurs orthographiques de la capacité à corriger les erreurs détectées. Cela consiste à trouver les mots du dictionnaire (ou lexique) qui sont similaires d'une certaine façon au mot mal orthographié. La tâche d'un correcteur orthographique se compose donc de trois sous-tâches : détecter les erreurs, générer les corrections possibles et classer les corrections suggérées. Pour y arriver, une variété de techniques fut inventée. Chacune d'elles est apparentée soit à la correction de mot inconnu, soit à la correction de mot mal placé, ou aux deux à la fois. Les erreurs d'orthographe peuvent être d'ordre typographique, cognitif ou phonétique. Les erreurs typographiques interviennent lorsque les touches du clavier sont appuyées dans le mauvais ordre (exemple : mian au lieu de main). Les erreurs cognitives résultent de l'ignorance de la bonne orthographe du mot (exemple : sécretaire au lieu de secrétaire). Les erreurs phonétiques constituent des cas d'erreurs cognitives. Une erreur phonétique fait référence à un mot erroné qui se prononce de la même façon que le mot correct (exemple : apeler / appeler). Les taux d'erreurs d'orthographe dans les textes dactylographiés sont entre 1 et 3% (Grudin, 1983). (Damerau, 1964) précise que 80% de ces erreurs sont de l'un des types suivants:

- Insertion d'une lettre supplémentaire
- Absence d'une lettre (suppression)
- Substitution d'une lettre par une autre
- Permutation de deux lettres.

La distance minimum d'édition ou simplement la distance d'édition est jusqu'aujourd'hui la technique la plus utilisée dans la correction des erreurs d'orthographe. Elle a été appliquée dans presque toutes les fonctions de correction orthographique dont les éditeurs de texte et les interfaces de langage de commande. Le premier algorithme de correction d'orthographe à base de cette technique était réalisé par (Damerau, 1964). Presque à la même période, Levenshtein développa aussi un algorithme similaire et qui semble être le plus utilisé. Plusieurs autres algorithmes sur la distance d'édition virent le jour par la suite. La distance d'édition est définie par Wagner comme étant le nombre minimum d'opérations d'édition requises pour transformer un mot en un autre (Kukich, 1992). Ces opérations sont l'insertion, la suppression, la substitution et la transposition.

Dans la plupart des cas, la correction d'une erreur d'orthographe nécessite l'insertion, la suppression ou la substitution d'une seule lettre ou la transposition de deux lettres. Quand un mot erroné peut être transformé en un mot du dictionnaire par l'inversion d'une de ces opérations, le mot du dictionnaire est considéré comme une correction plausible. Afin de réduire le temps de recherche, on utilise la technique de la distance d'édition inversée.Une autre approche pour réduire le nombre de comparaisons consiste à trier ou à partitionner le dictionnaire selon certains critères (ordre alphabétique, longueur des mots, occurrence des mots). Beaucoup d'autres techniques sont également utilisées dans la correction d'erreurs comme : la clé de similarité, les systèmes de règles, les techniques basées sur les n-grammes, les techniques probabilistes et les réseaux de neurones.

Cependant, La technique la plus largement utilisée dans la correction demeure la distance d'édition (Hsuan, 2008). Elle a une complexité en temps de $O(n \times m)$, avec n et m les tailles respectives des deux mots à comparer. Une technique développée par (Horst, 1993) alliant automate et distance d'édition a été utilisée pour une recherche rapide du mot correct le plus proche d'un mot erroné. Elle a une complexité en temps linéaire par rapport à la longueur du mot erroné, indépendamment de la taille du dictionnaire. Mais la complexité en espace de la méthode de (Horst, 1993) est exponentielle ($O\left(3.\exp\left(\sum_{i=1}^{N}|A_i|\right)\right)$, les A_i étant les mots du dictionnaire).

3 Etat des lieux sur l'informatisation du haoussa

Le haoussa (s'écrit aussi hausa ou hawsa) fait partie de la famille des langues afro-asiatiques. Il appartient au groupe des langues tchadiques (sous-groupe des langues tchadiques occidentales). Comparé aux autres langues africaines, le

haoussa est remarquablement unitaire. On distingue le haoussa standard (dialecte de Kano) du dialecte de l'ouest (Sokoto), des dialectes nigériens (Tibiri, Dogondoutchi, Filingué) et bien d'autres (http://www.humnet.ucla.edu/humnet/aflang/Hausa/hausa.html). Du point de vue vocalique, les mots haoussa supportent des tons hauts et des tons bas et on y observe une flexion de genre et de nombre (Mijinguini, Naroua, 2012). Géographiquement, le haoussa est la deuxième langue la plus parlée en Afrique et la plus répandue en Afrique noire avec environ cent millions de locuteurs à travers le monde. Le haoussa est aujourd'hui diffusé par les grandes stations radiophoniques du monde telles que VOA (États-Unis), BBC (Grande-Bretagne), CRI (Chine), RFI (France), IRIB (Iran), Deutsche Welle (Allemagne), Radio Moscou (Russie). Au Niger, le haoussa et les autres langues nationales sont utilisées par les médias nationaux, régionaux et locaux, publics comme privés (Maman, Seydou, 2010). Sur le plan cinématographique, l'industrie de vidéo en langue haoussa a connu un progrès remarquable. En effet, ce sont plus de 1000 films haoussa qui sont produits chaque année dont la quasi-totalité vient du Nigéria. La présence du haoussa sur Internet est très précaire. C'est malheureusement le cas de toutes les langues africaines malgré que celles-ci représentent 30% des langues du monde (Van Der, Gilles-Maurice, 2003). Le célèbre moteur de recherche Google, le navigateur Mozilla Firefox et bien d'autres logiciels et gadgets électroniques (téléphones mobiles notamment) disposent aujourd'hui d'une interface en langue haoussa. L'identifiant ISO de la langue haoussa est hau ou ha (ISO 639-3 et ISO 639-1). Sur le plan académique, les premiers poèmes composés en haoussa, écrits en alphabet arabe adapté à la notation des langues africaines (`ajami), datent du début du XIXe siècle. À cette époque également prend naissance une tradition de chroniques versifiées en haoussa, dont la plus connue est la Chronique de Kano (notée également en `ajami). À cette tradition s'est ajoutée dans les années 1930, à la suite de la colonisation britannique, une production littéraire en alphabet latin (pièces de théâtre, contes, nouvelles, romans, poésie) (Bernard, 2000). La langue haoussa est aujourd'hui enseignée dans des universités africaines et occidentales (Niger, Nigeria, Libye, Inalco (Paris), Université de Boston, UCLA). Le haoussa écrit est essentiellement fondé sur le dialecte de Kano et il existe deux systèmes d'écriture, l'un basé sur l'alphabet arabe (`Ajami), et l'autre utilisant l'alphabet latin (Boko) comme le montre la Figure 1. On remarque, dans le cas du Boko, la présence de 4 caractères spéciaux supplémentaires comme consonnes (ɓ, ɗ, ƙ et ƴ) et l'arrêt glottale (').

ب	[b]	م	[m]
ɓ	[ɓ]	ن	[n]
ƴ	[tʃ]	ر	[r], [ɾ]
د	[d]	س	[s]
ɗ	[ɗ]	ش	[ʃ]
ف	[β]	ت	[t]
غ	[g]	ts'	[ts']
ه	[h]	و	[w]
ج	[dʒ]	ى	[j]
ك	[k]	ع	[ʔ]
ƙ	[k']	ز	[z]
ل	[l]		

(a) Ajami

A a	[a], [æ]	M m	[m]
B b	[b]	N n	[n]
Ɓ ɓ	[ɓ]	O o	[o]
C c	[tʃ]	R r	[r], [ɾ]
D d	[d]	S s	[s]
Ɗ ɗ	[ɗ]	Sh sh	[ʃ]
F f	[β]	T t	[t]
G g	[g]	Ts ts	[ts']
H h	[h]	U u	[u], [u:]
I i	[i]	W w	[w]
J j	[dʒ]	Y y	[j]
K k	[k]	Ƴ ƴ	[ʔʲ]
Ƙ ƙ	[k']	Z z	[z]
L l	[l]	'	[ʔ]

(b) Boko

FIGURE 1 : Systèmes d'écriture du haoussa

La transcription latine, introduite par les Anglais au Nigeria au début du 20ème siècle, s'est imposée en 1930 comme orthographe officielle (http://www.humnet.ucla.edu/humnet/aflang/Hausa/hausa.html). Au Niger, il a fallu attendre 1981 pour rendre officielle une orthographe du haoussa utilisant l'alphabet latin. Cet alphabet fut complété en 1999 par un arrêté ministériel. Il s'agit du même alphabet que celui de la Figure 1 (b) auquel sont ajoutés les digraphes fy, gw, kw, ky, ƙw et ƙy représentant des sons spécifiques et considérés comme des consonnes. Le même arrêté définit les symboles du Tableau 1 pour la ponctuation.

Nom du symbole	Graphie
Point	.
Virgule	,
Point virgule	;
Deux points	:
Point d'interrogation	?
Point d'exclamation	!
Parenthèses	()
Guillemets	"
Trait d'union	-
Points de suspension	...
Tiret, à la ligne	-
Astérisque	*
Tirets ou parenthèses	...-...-
Tiret	-

TABLEAU 1 : Ponctuation officielle du texte haoussa au Niger

Le Boko est devenu la convention d'écriture dominante pour les documents scientifiques et éducatifs, les mass-médias, l'information et la communication générale depuis la deuxième moitié du 20ème siècle (Ahmed, 2009). Les ressources linguistiques constituent la première étape dans l'informatisation d'une langue (Chanard, Popescu-Belis, 2001). C'est grâce à elles qu'il peut être possible de concevoir les outils informatiques (éditeurs, correcteurs d'orthographe et de grammaire, dictionnaire électronique ...) adaptés à la langue et d'assurer sa présence dans le cyberespace. Mais ces ressources sont rares pour les langues africaines. C'est ainsi que des projets et études déjà réalisés ou en cours visent la constitution ou l'exploitation de ces ressources linguistiques pour une informatisation totale des langues africaines. Par exemple, le projet PAL vise l'adaptation des TIC aux langues africaines afin de les rendre plus accessibles aux populations autochtones (Don, 2011). Bien que le premier travail sur la lexicographie du haoussa moderne date de longtemps, le dictionnaire de (Bargery, 1934) parait être le plus important et le plus large (avec 39000 mots). Il est bilingue haoussa-anglais et renferme une section de vocabulaire anglais-haoussa. Dans leur genèse de la lexicographie du haoussa, (Roxana, Paul, 2001) mentionnent plusieurs autres dictionnaires avant d'évoquer le dictionnaire bilingue haoussa-français du linguiste nigérien et natif haoussa (Minjinguini, 2003). Ce dictionnaire est, selon eux, « la plus récente référence scientifique en lexicographie du haoussa ». Il comprend 10000 entrées bien illustrées et se base largement sur le haoussa standard du Niger, constitué essentiellement du dialecte de Damagaram au lieu de celui de Kano qui dominait dans toutes les recherches lexicographiques précédentes. Rappelons de passage que (Paul, 2000) est l'auteur de l'œuvre la plus complète sur la grammaire moderne du haoussa. La majorité des langues bien dotées disposent de corpus bien formés. Ce qui n'est pas le cas pour les langues africaines. Les recherches actuelles sur ces langues choisissent comme alternative transitoire des corpus écrits et oraux. Une autre alternative pour les langues africaines consiste à constituer des corpus à partir du Web (Gilles-Maurice, 2002).

L'entrée ou la saisie de texte est une autre difficulté à surmonter dans l'informatisation du haoussa et des langues africaines. En effet, les claviers compatibles à ces langues ne sont pas encore mis au point. Saisir certains caractères haoussa sur un clavier demande aujourd'hui une certaine acrobatie. La solution pour contourner cette difficulté consiste en l'utilisation de claviers virtuels permettant d'écrire tous les caractères des langues africaines. Une évaluation (Enguehard, Naroua, 2008) de ce genre de claviers concernant 5 langues du Niger (Fulfulde, Haoussa, Kanuri, Songhai-Zarma, Tamashek) a conclu sur la recommandation, pour ces langues, du clavier virtuel du laboratoire LLACAN.

Les logiciels de traitement de texte tels que MS Word et OpenOffice.org Writer peuvent être utilisés pour la correction d'un texte écrit en haoussa grâce à la constitution de dictionnaire utilisateur. Cependant, toutes les méthodes existantes restent limitées et inadéquates dans le cas des langues africaines ; d'où le besoin de concevoir des correcteurs orthographiques adaptés à ces langues (Enguehard, Mbodj, 2004) . Malgré la rareté des ressources linguistiques, il est bien possible de mettre au point ces correcteurs quitte à les améliorer dans le temps.

Certains logiciels populaires (MS Word, OpenOffice.org Writer, Firefox, etc.) offrant la possibilité de leur créer des extensions (add-ons, plugins), il serait avantageux de concevoir des correcteurs d'orthographe pouvant facilement leur être intégrés.

4 Conception et réalisation d'un correcteur orthographique pour le haoussa

Après synthèse des techniques de détection et de correction d'erreurs d'orthographe, la présentation de la langue haoussa et le point sur son informatisation, nous nous attelons à la conception et la réalisation d'un correcteur orthographique pour le haoussa. Nous en exposons les approches et techniques choisies ainsi que les détails d'implémentation de la solution proposée.

4.1 Choix techniques

Dans cette section, nous présentons les structures de données utilisées pour la conception du correcteur ainsi que les procédures nécessaires pour la détection et la correction d'erreurs en haoussa. Nous avons opté pour l'approche de la conception objet avec un langage algorithmique inspiré de Java (Christophe, 2008). Ce sera sans nous attarder sur la théorie des concepts sous-jacents tels que : classe, objet, méthode, attribut, instance, etc. (Brett et al., 2006) et (Christophe, 2008) sont de bonnes références à ce sujet. Au vu des ressources linguistiques à notre disposition, une technique basée sur un dictionnaire nous semble la plus adaptée pour la conception du correcteur haoussa. Pour le choix du dictionnaire, nous avons opté pour celui de (Mijinguini, 2003). D'abord parce qu'il nous est accessible, ensuite en raison de ses atouts. Le dictionnaire contient tous les mots (y compris les inflexions et les dérivations). Il est stocké en mémoire secondaire sous forme de fichier texte. L'encodage de caractères étant évidemment UTF-8. La détection d'erreurs se fait indépendamment du contexte. Un mot erroné est identifié par une simple recherche dans le dictionnaire. Pour implanter le dictionnaire en mémoire, nous utilisons soit une table de hachage soit un trie. L'implémentation doit permettre au moins les primitives suivantes :

- Ajouter un mot au dictionnaire (méthode add)
- Vérifier si un mot se trouve dans le dictionnaire (méthode contains)
- Supprimer un mot du dictionnaire (méthode remove)

Chaque nœud du trie a autant de liens qu'il y a de caractères dans l'alphabet et ces derniers sont implicitement stockés dans la structure de données. A chaque chaîne de caractères valide correspond une valeur. Celle-ci peut être de tout type. Elle peut être exploitée ici pour stocker des informations (définition, classe grammaticale, traduction vers une autre langue, etc.) sur chaque mot du dictionnaire.

En notation objet le trie se présente comme le montre la classe Trie de la Figure 2. Chaque nœud du trie est représenté par la structure de données Node.

L'attribut R de la classe Trie correspond au nombre des symboles ou lettres de l'alphabet. Les digraphes de l'alphabet haoussa du Niger n'étant pas codés comme un seul caractère, nous ne considérerons que les monographes, soit un total de 28 lettres. A ces lettres nous ajoutons le tiret ('-') (code Unicode \u002D) afin de pouvoir stocker les mots composés. Pour une langue supportée par le code ASCII, il n'est pas nécessaire d'avoir un attribut alphabet pour la classe Trie, les caractères étant représentés par des entiers consécutifs de 0 à 127 donc par les indices du tableau next (Node[]). Ce qui n'est pas le cas du haoussa où les lettres ont des points de code dans les plages suivantes :

- Majuscules : 39, 65 à 80, 82 à 85, 87 à 90, 385, 394, 408, 435.
- Minuscules : 97 à 112, 114 à 117, 119 à 122, 595, 599, 409, 436.

Représenter les caractères par les indices du tableau next conduira à prendre 599 comme valeur de R au lieu de 56 (28x2), ce qui conduit à un gaspillage d'espace mémoire (parce qu'occupée par des liens inutiles) et des vérifications supplémentaires pour éviter que des mots étrangers ne soient ajoutés au trie. Pour éviter ce problème, une astuce (Robert, Kevin, 2011) consiste à trouver une fonction de correspondance entre les indices du tableau next et les lettres de l'alphabet. C'est la raison de la présence de l'attribut alphabet de la classe Trie. Il est ici de type String mais il peut bien être un tableau de caractères. Deux méthodes supplémentaires effectuent la correspondance : toChar pour retrouver le caractère correspondant à un indice donné et toIndex pour convertir un caractère donné en indice. Les méthodes

charAt et indexOf de la classe String peuvent efficacement être utilisées. Et pour rendre l'astuce plus flexible, nous pouvons carrément déléguer cette tâche à une interface Alphabet qui définirait toChar et toIndex. La méthode keysThatMatch est très intéressante. En effet, elle permet de rechercher dans le trie les mots qui répondent à un motif donné. Les motifs utilisés ici sont ceux avec un caractère de remplacement (wildcard), par exemple un point ('.'). Ainsi avec un motif comme '.ada' cette méthode va renvoyer les mots du dictionnaire qui sont constitués d'une lettre (quelconque) suivie du suffixe 'ada' : dada, fada, kada, lada, tada, wada. C'est cette possibilité que nous exploitons pour mettre en œuvre la distance d'édition inversée. La méthode keysThatMatch utilise une structure de données List pour conserver les résultats de la recherche. La classe List dispose de méthodes pour ajouter un élément, pour vérifier l'existence d'un élément et pour supprimer un élément.

Pour faire abstraction de l'implantation du dictionnaire réel, ajouter de la flexibilité, simplifier la maintenance et faciliter l'évolutivité du correcteur, le dictionnaire abstrait est représenté par une classe (TrieBasedDico ou HashBasedDico) qui implémente une interface (ou une classe abstraite) Dico. Celle-ci définit les méthodes (add, remove, contains) nécessaires pour opérer sur un dictionnaire. Les classes TrieBasedDico et HashBasedDico sont conçues par composition à partir respectivement de la classe Trie et de la classe HashSet.

La liste des mots candidats pour la correction d'un mot erroné est déterminée par plusieurs étapes que nous décrivons ici. Une fois qu'un mot est identifié comme étant erroné, on procède à la détermination de la forme de l'erreur. Nous avons défini trois types d'erreurs (inspiration venue de nos recherches sur OpenOffice.org) :

- IS_NEGATIVE_WORD : Erreur causée par la présence, dans le mot, d'un chiffre ou d'un caractère étranger à l'alphabet (par exemple x, v, q, etc.). Le mot est alors qualifié de négatif.
- CAPTION_ERROR : Erreur de casse. C'est lorsqu'un mot qui devait être écrit avec la première lettre en majuscule est écrit tout en minuscule.
- SPELLING_ERROR : représente toutes les autres formes d'erreurs d'orthographe.

Les types d'erreurs sont de type entier court encapsulés comme champs statiques dans la classe LySpellFailure. Le correcteur dispose de deux méthodes pour la détermination des erreurs. D'abord la méthode getSpellFailure, qui analyse un mot donné, renvoie -1 si le mot est correct ou, dans le cas contraire, un des trois types d'erreur cités ci-haut. Ensuite la méthode isValid qui vérifie si un mot donné est valide en fonction du résultat renvoyé par getSpellFailure et des paramètres de la correction orthographique. Si getSpellFailure renvoie une valeur :

- égale à -1, le mot est valide et isValid renvoie true
- différente de -1, les paramètres de correction sont pris en compte pour déterminer la validité. Par exemple lorsqu'on choisit de ne pas corriger les mots avec chiffres et que le mot à corriger contient des chiffres, isValid renvoie true. Cette méthode peut être exploitée pour corriger l'orthographe au cours de la frappe.

L'objet currentLanguage représente la langue en cours de prise en charge par le correcteur. Il est une instance de la classe Language. La recherche des suggestions de correction est déléguée à proposer, une instance d'une classe qui implémente l'interface Proposer.

La méthode getProposals fournit les suggestions de correction pour un mot invalidé par isValid et ce en fonction du type d'erreur détectée par getSpellFailure.

1) Exploitation des caractéristiques de l'alphabet

La langue traitée est représentée par la classe Language. Après plusieurs tentatives, nous avons préféré que le dictionnaire soit un attribut de la langue et non l'inverse. L'attribut locale de la classe Language stocke les informations sur la langue traitée. Il est de type Locale et fournit par exemple le code 2 lettres ISO 639-1 de la langue, le code 2 lettres ISO 3166 du pays ainsi que les noms complets de la langue et du pays. Ce qui correspond respectivement à ha, NE, haoussa (Niger) pour le haoussa du Niger. Nous exploitons ces données pour le nommage des ressources et pour l'affichage destiné à l'utilisateur. L'attribut properties qui est de type Map regroupe d'autres propriétés pour la langue que nous utilisons pour la conception du correcteur et qui ne sont pas fournies par Locale. Il s'agit pour le moment de l'alphabet de la langue, des caractères spéciaux de l'alphabet, des caractères ressemblant aux caractères spéciaux et les signes de ponctuation que nous avons repartis en deux propriétés : les séparateurs de mots et les signes de fin de phrase. Tous les caractères de l'alphabet sont fournis sous forme de codes Unicode. La classe chargée de trouver les suggestions implémente l'interface Proposer qui définit deux méthodes : isNegativeWord et propose. Les méthodes des classes

TrieBasedDicoProposer et HashBasedDicoProposer utilisent en partie les caractéristiques de l'alphabet dans la recherche de suggestions.

2) Utilisation de la distance d'édition inversée pour trouver les mots à suggérer

La recherche des mots candidats pour la correction est faite grâce à la distance d'édition inversée comme suit :

- Tous les mots ayant une distance d'édition égale à 1 avec le mot erroné sont générés par application des opérations d'édition que sont l'insertion, la suppression, la substitution et la transposition.
- Chaque mot généré précédemment est recherché dans le trie ou la table de hachage. S'il y est, alors il est retenu comme une correction possible du mot erroné.

La recherche est effectuée par la méthode privée proposeByReverseEditDistance. Cette méthode est basée en réalité sur la méthode keysThatMatch. Elle prend en argument un paramètre de type TrieBasedDico et un mot ou un pattern et renvoie le résultat sous forme d'un tableau de Strings. Une méthode similaire est conçue dans le cas de la table de hachage. Les méthodes qui permettent d'appliquer les opérations d'édition à un mot donné sont fournies par la classe StringTools. Celle-ci regroupe un nombre d'outils à usage partagé entre les différentes classes.

3) Utilisation de la distance d'édition pour ordonner les suggestions

La distance minimum d'édition est utilisée pour classer les mots candidats. Ceux qui sont les plus proches du mot erroné sont placés en tête de la liste proposée. Pour mettre cela en œuvre, un comparateur a été conçu. Le diagramme de classe général de la conception que nous venons de décrire est donné par la Figure 2 ci-dessous :

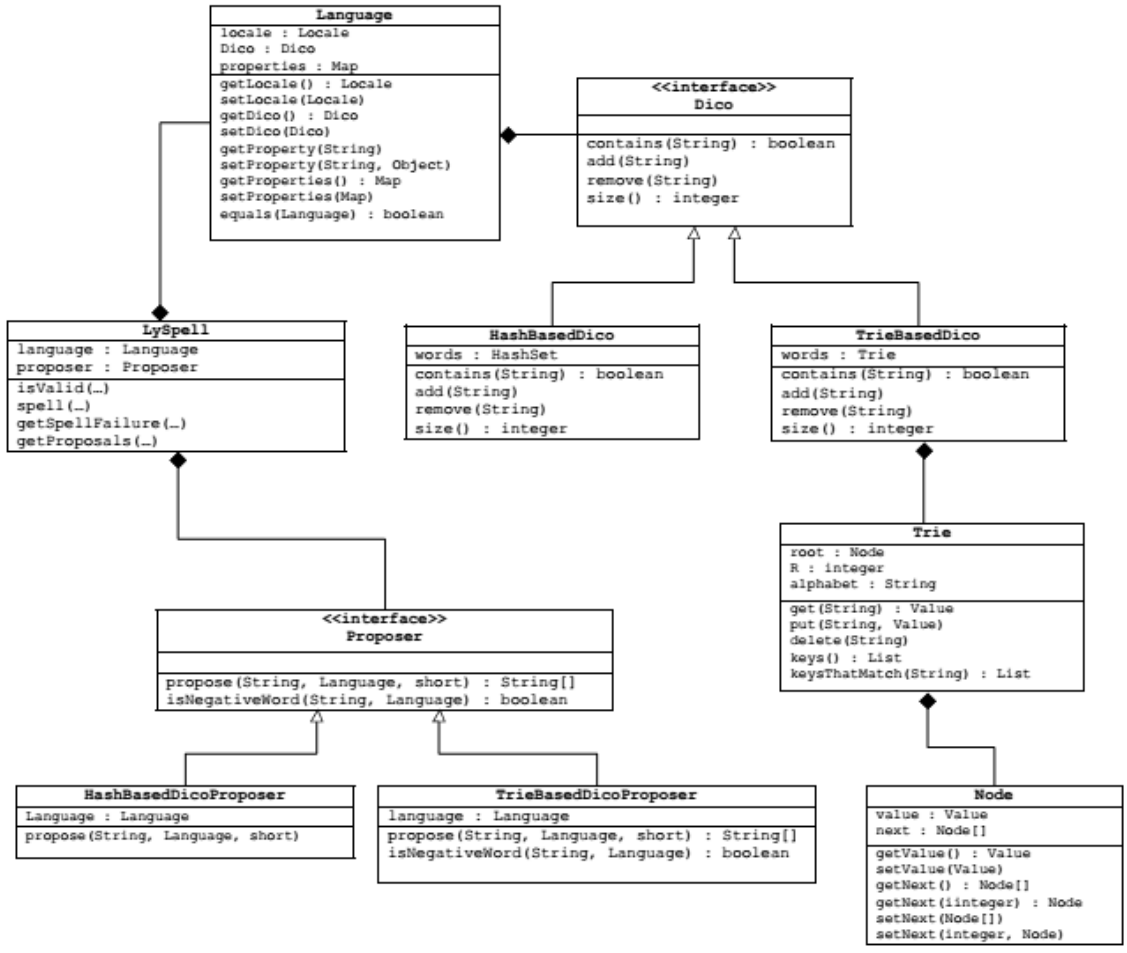

FIGURE 2 : Diagramme de classe global

4.2 Codage, déploiement et test du correcteur

Pour le codage et le développement du correcteur nous avons opté pour le langage Java et l'IDE NetBeans. Deux versions ont été développées.

a) Version autonome : LyTexEditor et LySpell

Elle comprend un éditeur de texte LyTextEditor qui intègre LySpell, le correcteur que nous avons conçu. LyTextEditor a été conçu dans un premier temps pour les besoins de développement et test du correcteur indépendamment des contraintes d'intégration à d'autres éditeurs. Il donne les possibilités suivantes :

- saisir un texte
- ouvrir un fichier texte existant
- corriger un texte avec LySpell
- sauvegarder un texte.

La correction orthographique est accessible via le menu Outils ou par la touche F7. Par exemple, la Figure 3 montre la boite de dialogue pour la correction interactive de l'orthographe.

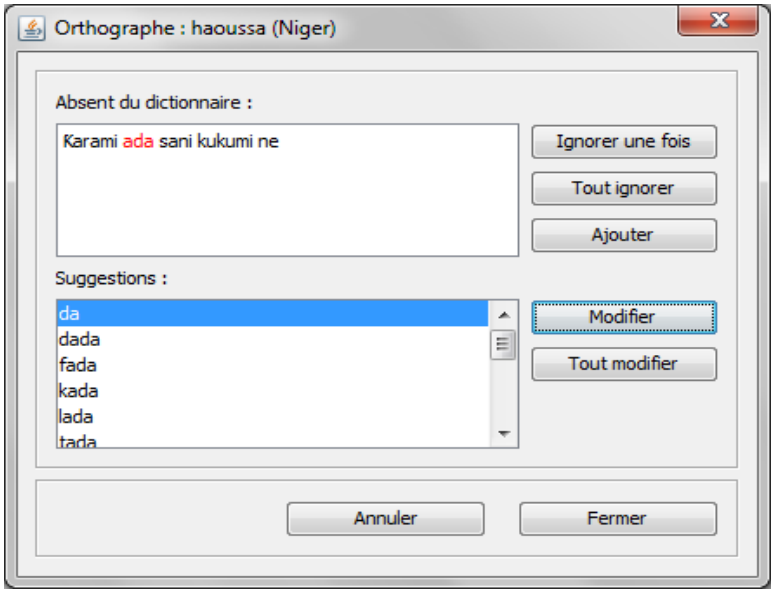

FIGURE 3 : Boîte de dialogue pour la correction de l'orthographe avec LySpell dans LyTextEditor

b) Version add-on pour OpenOffice.org

Après plusieurs recherches et avec l'aide de l'OpenOffice.org Developer's Guide, nous avons pu développer l'add-on. Comme OpenOffice.org 3 n'intègre pas le haoussa du Niger, obligation est faite de choisir l'option haoussa du Nigéria ou celui du Ghana. La Figure 4 ci-dessous montre l'utilisation de LySpell dans OpenOffice.org 3 Writer.

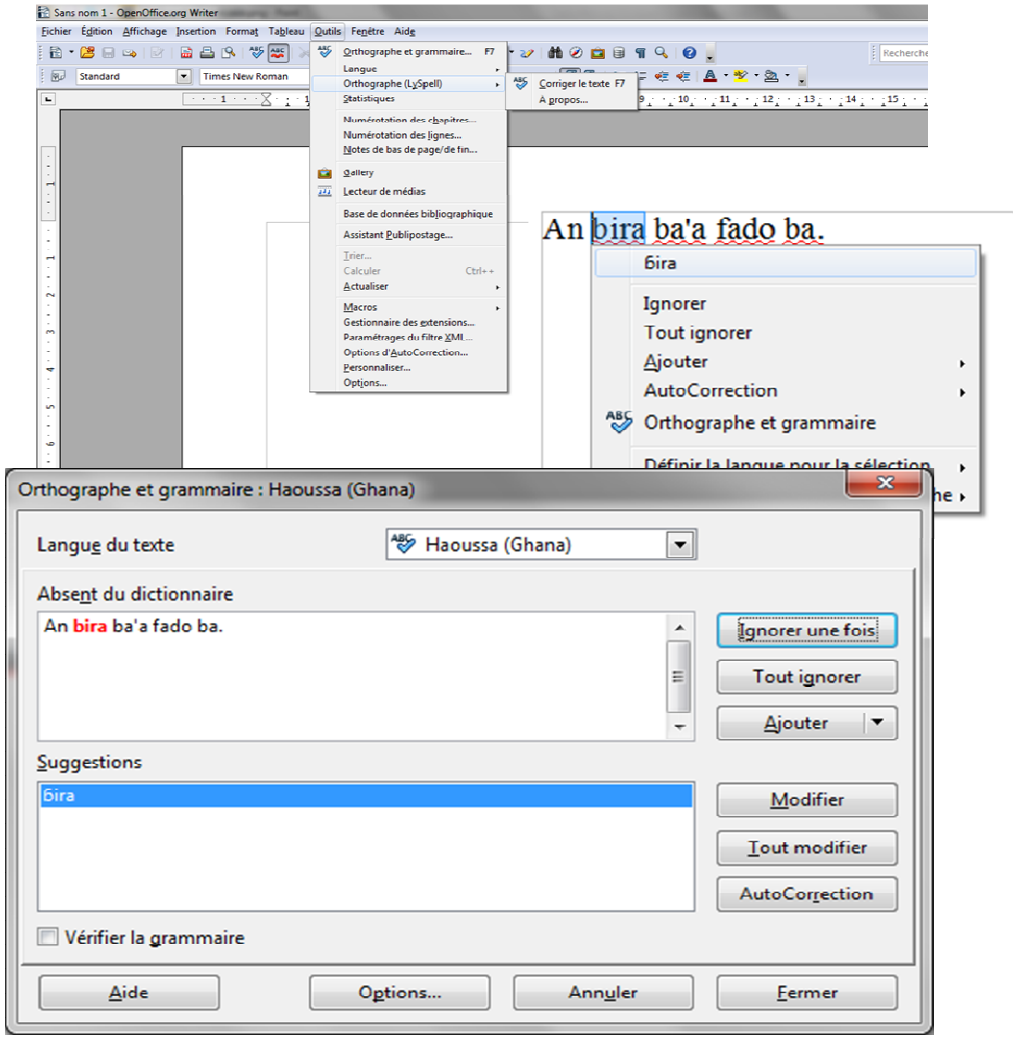

FIGURE 4 : Correction de l'orthographe avec LySpell dans OpenOffice.org

Avec la portabilité de Java, LyTextEditor et LySpell peuvent être utilisés normalement sur toutes les plateformes. LySpell offre également la possibilité de prendre en charge, sans besoin de toucher au code, la correction d'orthographe pour d'autres langues. Il suffit pour cela de fournir les fichiers nécessaires à savoir le dictionnaire et l'alphabet.

5 Conclusion et perspectives

Dans ce travail, nous avons conçu et développé un correcteur orthographique pour la langue haoussa. Ledit correcteur a été testé en tant que programme autonome à travers un éditeur de texte conçu à cet effet et en tant qu'extension pour la suite bureautique OpenOffice.org. Ces résultats montrent qu'il est bien possible de mettre à profit les techniques prouvées et les ressources linguistiques disponibles pour concevoir des outils de traitement automatique pour les langues africaines en général et pour le haoussa en particulier. Ils confirment aussi que les structures de données trie et table de hachage offrent de meilleurs performances pour stocker un dictionnaire. Cependant, les possibilités et les résultats offerts par la structure de données trie sont nettement meilleurs à ceux de la table de hachage. Il faut noter que lorsque le nombre de wildcard est supérieur à 1, seul le trie donne, sans grande gymnastique, un résultat satisfaisant. Par exemple, pour le mot incorrect "zurmakakke" et lorsque le dictionnaire est implanté par un trie, on obtient la suggestion "zurmaƙaƙƙe". Par contre, aucune suggestion n'est obtenue dans le cas de la table de hachage.

Le correcteur LySpell résultant de cette étude exploite comme seules ressources linguistiques le dictionnaire et l'alphabet de la langue haoussa. Il a cependant été pensé de façon qu'il puisse aussi servir pour d'autres dialectes haoussa et d'autres langues.

Malgré que nous n'ayons pas pu effectuer tous les tests nécessaires sur les performances de LySpell, nous osons espérer que les résultats auxquels nous avons abouti apporteront une valeur ajoutée à l'informatisation du haoussa et contribueront à son utilisation effective dans les institutions d'enseignement et les médias.

Pour améliorer les performances du correcteur ici conçu, il peut être envisagé dans des futurs travaux de :

- Exploiter les règles de la morphologie de la langue haoussa. Cela aura un triple avantage. D'abord la taille du dictionnaire en mémoire sera considérablement réduite. Ensuite les suggestions de correction pourraient être plus précises. Enfin, il serait ainsi possible de créer un correcteur orienté Hunspell pouvant être intégré facilement et plus adéquatement à un large éventail de programmes à commencer par OpenOffice.org.
- Renforcer la correction orthographique du haoussa en y ajoutant la prise en charge de la grammaire.

Références

AHMED N. (2009). Adaptation des écritures et de la lecture des langues étrangères au pays Haoussa de l'Afrique de l'Ouest. *Synergies Algérie n°6 – 2009*, 61-69.

AHO A. V., CORASICK M. J. (1975). Efficient String Matching: An Aid to Bibliographic Search. *Communications of the ACM*, 18 (6), 333-340.

BARGERY G.P. (1934). A Hausa-English Dictionary and English-Hausa Vocabulary. *Oxford University Press*, London.

BERNARD C. (2000). Les langues au Nigeria. *Notre Librairie, Revue des littératures du Sud, Littératures du Nigéria et du Ghana*, 2, (141), 8-15.

BRETT M., Gary P., David W. (2006). Head First Object-Oriented Analysis and Design. *O'Reilly*.

CHANARD C., POPESCU-BELIS A. (2001). Encodage informatique multilingue : application au contexte du Niger. *Les Cahiers du Rifal*, 22, 33-45.

CHRISTOPHE D. (2008). Apprendre à programmer, algorithmes et conception objet. *2e ed., Eyrolles*.

CYRIL N. A. (1967). String similarity and misspellings. *Communications of the A.C.M.*, 10, (5), 302-313.

DAMERAU F.J. (1964). A technique for computer detection and correction of spelling errors. *Comm.* ACM 7, 3, 171-176.

DANIEL J., JAMES H. M. (2000). Speech and Language Processing. *Prentice Hall, Englewood Cliffs, Inc*.

DON O. (2011). Les langues africaines a l'ère du numerique, défis et opportunités de l'informatisation des langues autochtones. *Les Presses de l'Université Laval, CRDI*.

ENGUEHARD C., NAROUA H. (2008). Evaluation of Virtual Keyboards for West-African Languages. Proceedings of the *Sixth International Conference on Language Resources and Evaluation (LREC'08)*, Marrakech, Morocco, 28-30.

ENGUEHARD C., MBODJ C. (2004). Des correcteurs orthographiques pour les langues africaines. *Bulletin de Linguistique Appliquée et Générale.*

ENGUEHARD C., SOUMANA K., MATHIEU M., ISSOUF M., MAMADOU L. S. (2011). "Vers l'informatisation de quelques langues d'Afrique de l'Ouest", 4ème atelier international sur l'Amazighe et les Nouvelles Technologies, IRCAM, Rabat, Maroc.

GILLES-MAURICE D. S. (2002). Web for/as Corpus: A Perspective for the African Languages. *Nordic Journal of African Studies*, 11 (2), 266-282.

GRUDIN J. T. (1983). Error patterns in novice and skilled transcription typing. *In Cooper W. E. (Ed.). Cognitive Aspects of Skilled Typewriting, Springer-Verlag*, New York, 121-139.

HORST B. (1993). A Fast Algorithm for Finding the Nearest Neighbor of a Word in a Dictionary. *IAM-93-025*.

HSUAN L. L. (2008). Spell Checkers and Correctors: a unified treatment. *Master dissertation*.

KNUTH D. (1973). The Art of Computer Programming. *Addison-Wesley Publishing Co.*, Philippines, 3.

KUKICH K. (1992). Techniques for automatically correcting words in text. *ACM Computing Surveys*, 24 (4).

LEHAL G. S., SINGH K. (2000). A Comparative Study of Data Structures for Punjabi Dictionary. *5th International Conference on Cognitive Systems, reviews & previews, ICCS'99*, 489-497.

MAMAN M. G., SEYDOU H. H. (2010). Les Langues de scolarisation dans l'enseignement fondamental en Afrique subsaharienne francophone : cas du Niger. *Rapport d'étude pays*.

MARK P. N. (2009). A Comparison of Dictionary Implementations.

MINJINGUINI A. (2003). Dictionnaire élémentaire hausa-français. *les éditions GG*.

MIJIGUIN A., NAROUA H. (2012). Règles de formation des noms en haoussa. Actes de la *conférence conjointe JEP-TALN-RECITAL 2012, Atelier TALAf 2012: Traitement Automatique des Langues Africaines*, 63-74.

PAUL N. (2000). The Hausa Language An Encyclopedic Reference Grammar. *Yale University Press*, New Haven.
PETERSON J. L. (1980). Computer Programs for Detecting and Correcting Spelling Errors. *Comm. ACM*, 23 (12).
PIERRE M. N. (2006). An introduction to language processing with Perl and Prolog. *Springer-Verlag Berlin Heidelberg*, 2-3.
ROBERT S., KEVIN W. (2011). Algorithms. *4e ed., Addison Wisley*.
ROXANA M. N., PAUL N. (2001). The Hausa Lexicographic Tradition. *Lexikos11, AFRILEX-reeks, series*, 11, 263-286.
SUZAN V. (2002). Context-sensitive spell checking based on word trigram probabilities. *Master thesis*.
VAN DER A. V., GILLES-MAURICE D. S. (2003). The African Languages on the Internet: Case Studies for Hausa, Somali, Lingala and isiXhosa. *Cahiers Du Rifal*, 23, 33–45.