RAPPORT DE STAGE DE L3

12 septembre 2012

Analyse des paramètres pour l'amélioration d'un algorithme à colonies de fourmis appliqué à la désambiguïsation de textes

Yann DUPLOUY
yduplouy@ens-cachan.fr

Stage effectué dans l'équipe GETALP du LIG

Sous l'encadrement de :

Didier SCHWAB
Didier.Schwab@imag.fr

Jérôme GOULIAN
Jerome.Goulian@imag.fr

INTRODUCTION

J'ai effectué mon stage dans l'équipe GETALP [1] (Groupe d'Étude pour la Traduction/le Traitement Automatique des Langues et de la Parole) du Laboratoire d'Informatique de Grenoble [2], sous la direction de Didier SCHWAB et Jérôme GOULIAN. Comme son nom l'indique, cette équipe s'intéresse aux divers aspects du traitement automatique des langues naturelles. Mes encadrants travaillent plus particulièrement sur la désambiguïsation de textes. Ce problème spécifique est facile pour un être humain qui comprendra, dans la majorité des cas, un texte sans efforts, en s'aidant du contexte. Mais la tâche est beaucoup moins intuitive pour une machine : il est difficile d'imaginer un algorithme permettant de résoudre ce problème.

TABLE DES MATIÈRES

Page

1 Traitement des langues et désambiguïsation **2**
- 1.1 Mesures de similarité . . . 2
  - 1.1.1 Mesures de similarité à base de traits . . . 2
  - 1.1.2 Mesures de similarité à base de distance taxonomique . . . 3
  - 1.1.3 Mesures de similarité à base de contenu informationnel . . . 4
- 1.2 Désambiguïsation lexicale . . . 5
  - 1.2.1 Algorithmes globaux . . . 5
  - 1.2.2 Recuit simulé . . . 5
  - 1.2.3 Algorithme génétique . . . 5
  - 1.2.4 Exploration pseudo-aléatoire de graphes : PageRank . . . 6
  - 1.2.5 Exploration pseudo-aléatoire de graphes : colonies de fourmis . . . 6

2 Analyse des paramètres d'un algorithme à colonies de fourmis **7**
- 2.1 Paramètres de l'algorithme . . . 8
- 2.2 Méthode d'analyse . . . 8
  - 2.2.1 Recuit simulé utilisé . . . 8
  - 2.2.2 Choix de configuration et corpus . . . 9
- 2.3 Résultats obtenus . . . 10
  - 2.3.1 Analyse des paramètres modifiant le comportement de l'algorithme . . . 10
  - 2.3.2 Choix des valeurs des paramètres pour améliorer le score obtenu . . . 10

A Recuit simulé **12**

1. http://getalp.imag.fr
2. http://www.liglab.fr

I — TRAITEMENT DES LANGUES ET DÉSAMBIGUÏSATION

Le domaine du traitement des langues date quasiment de l'apparition des premières machines : ce qu'on appelle aujourd'hui le *test de Turing* a été présenté par Alan TURING en 1950 [24]. Pour réussir ce test, la machine doit être capable de participer à une conversation en temps réel sans que l'humain qui lui parlerait soit capable de distinguer – avec le seul contenu de la conversation – s'il discute avec un autre être humain ou une machine. Un autre exemple montrant la difficulté du problème est la traduction automatique : la démonstration faite en 1954 par IBM sur la traduction totalement automatique de 60 phrases du russe vers l'anglais était encourageante – ses auteurs pensant que dans un délai de cinq ans, ce problème allait être totalement résolu – il s'avère qu'aujourd'hui on constate facilement que ce n'est toujours pas le cas.

Le problème traité dans ce stage se pose lorsque l'on veut résoudre l'un des deux problèmes précédents : pour pouvoir traduire une phrase ou y répondre, il faut déjà la comprendre, et pour cela, il faut savoir quel est le sens de la phrase – ce qui peut être déduit du sens de chacun des mots présents dans la phrase. Les approches existantes – que l'on décrira succintement plus loin – reposent principalement sur des mesures de similarité : en effet, si deux mots apparaissent dans une même phrase, ces mots doivent avoir des sens suffisemment proches. Pour réaliser une désambiguïsation, il faut trouver un moyen d'indiquer à la machine à quel point ces mots sont proches.

I.1. MESURES DE SIMILARITÉ

La majorité des mesures de similarité présentées dans ce rapport se basent sur WordNet [14], un inventaire de sens et de relations entre ceux-ci, structuré autour de synsets. Un synset représente un sens de mot : il s'agit plus précisément d'un ensemble de *synonymes* autour d'un concept. Ces différents synsets sont reliés entre eux par des relations lexicales (comme l'antonymie) ou taxonomiques (hyperonymie [3], méronymie [4]. On ne présentera ici que des mesures de similarité se basant sur des connaissances.

I.1.1. MESURES DE SIMILARITÉ À BASE DE TRAITS

La notion de similarité sémantique a d'abord été traitée dans le domaine de la psychologie cognitive, en particulier par le travail mené par Tversky [25] qui calcule la similarité entre deux objets comme le nombre pondéré de propriétés en commun auxquelles on retire le nombre pondéré de propriétés spécifiques à chaque objet. Dans le cadre de la recherche de la similarité des mots, les objets sont les sens, et les propriétés des traits présents dans les sens : certains traits sont moins importants que d'autres, et auront une pondération plus faible. Cet aspect est exploité par Pirró et Euzenat [18], qui définissent une notation spécifique. Si on l'on note $\psi(s)$ l'ensemble des traits se rapportant à un sens s, et que l'on définit une fonction F associant une pertinence aux traits on peut définir la similarité de Tversky par :

$$\mathrm{sim}_{\mathrm{tvr}}(s_1, s_2) = \theta F(\psi(s_1) \cap \psi(s_2)) - \alpha F(\psi(s_1) \setminus \psi(s_2)) - \beta F(\psi(s_2) \setminus \psi(s_1))$$

où θ, α et β sont des facteurs marquant respectivement l'importance de la similarité entre les sens, et des dissimilarités entre s_1 et s_2 puis s_2 et s_1.

Lesk [12] a quant a lui proposé une mesure de similarité très simple : la mesure de similarité entre deux sens s_1 et s_2 est ici le nombre de mots en commun dans leurs définitions, sans prendre en compte l'ordre des mots dans ces définitions (on parle alors de *sac de mots*). Cette mesure présente l'avantage de ne nécessiter qu'un dictionnaire, et d'être très simple à calculer. Lesk avait développé un algorithme de désambiguïsation lexicale calculant de manière exhaustive tous les sens de tous les mots du contexte. Navigli [15] a créé une variante de l'algorithme, limitant ce calcul à une fenêtre autour du mot dont le sens est recherché. Néanmoins, cette mesure n'est pas très précise, compte tenu des définitions souvent trop concises et de l'absence de certains mots importants dans ces définitions.

Cette mesure a cependant donné lieu à diverses extensions : Wilks et Stevenson [26] ont proposé de pondérer les mots des définitions par la longueur de celle-ci, de sorte à tenter d'éviter de privilégier les définitions longues. Banerjee et Pedersen [1] ont proposé quant à eux un *Lesk étendu* qui prend en compte les relations taxonomiques présentes dans WordNet et qui calcule d'une autre manière le recouvrement entre les mots des définitions. Ils prennent en

3. Par exemple, «coiffure» est un hyperonyme de «chapeau» ou de «couronne». Plus généralement, un mot A est un hyperonyme de B si l'extension de A, plus général, englobe l'extension de B, plus générique

4. Par exemple, «bras» est un méronyme de «corps», ou «toit» un méronyme de «maison». Plus généralement, A est un méronyme de B si le signifié de A désigne une sous-partie du signifié de B.

compte les hyperonymes, les hyponymes [5], les méronymes, les holonymes [6] mais aussi d'autres relations présentes dans WordNet (*attribut*, *similaire à*, *voir aussi*). Pour garantir que la mesure soit symétrique, une paire de relation (R_1, R_2) n'est gardée que si (R_2, R_1) est elle aussi présente. Ils se servent par ailleurs de la loi de Zipf [28] mettant en évidence une relation quadratique entre la longueur d'une phrase, et sa fréquence d'occurence dans un corpus.

I.1.2. MESURES DE SIMILARITÉ À BASE DE DISTANCE TAXONOMIQUE

Le principe général de ces mesures est de compter le nombre d'arcs séparant deux sens dans un graphe taxonomique, formé à partir des relations taxonomiques (hyperonymie, méronymie, hyponymie ou hyperonymie). La relation de deux sens s_1 et s_2, par rapport à leur sens commun le plus spécifique s_3, dans une taxonomie peut être représentée par la figure suivante, issue du travail de Wu et Palmer en 1994 [27], qui nous servira à exprimer les formules des différentes mesures de similarité.

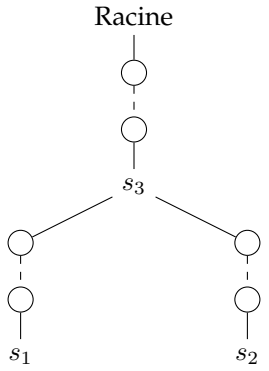

FIGURE 1 – Relation entre deux sens s_1 et s_2 et leur sens commun s_3 dans une taxonomie

Pour les mesures suivantes, N_1 correspond à la distance entre s_1 et s_3 dans la taxonomie, N_2 à la distance entre s_2 et s_3, et finalement, N_3 est la distance entre la racine et s_3.

La mesure de Rada [19] est la première à se servir de la distance entre les noeuds correspondant aux deux sens, sur les liens d'hyponymie et hyperonymie :

$$\mathrm{sim}_{\mathrm{Rada}}(s_1, s_2) = d(s_1, s_2) = N_1 + N_2$$

En se basant sur le fait que la profondeur d'un sens implique sa spécificité, et donc que deux sens profondément placés dans l'arbre taxonomique sont plus proches sémantiquement que des termes généraux, Wu et Palmer [27] ont proposé de prendre en compte la distance entre l'ancêtre commun le plus spécifique et la racine :

$$\mathrm{sim}_{\mathrm{WP}}(s_1, s_2) = \frac{2N_3}{N_1 + N_2 + 2N_3}$$

Une autre mesure basée sur la mesure de Rada est celle de Leacock et Chodorow [11] qui, au lieu de normaliser par la profondeur relative de la taxonomie par rapport aux sens, choisit de normaliser par rapport à la profondeur totale de la taxonomie (que l'on notera D) puis de normaliser par un logarithme :

$$\mathrm{sim}_{\mathrm{LCH}} = -\log\left(\frac{N_1 + N_2}{2D}\right)$$

I.1.3. MESURES DE SIMILARITÉ À BASE DE CONTENU INFORMATIONNEL

Resnik [20] propose une approche basée sur la détermination du contenu informationnel du concept commun le plus spécifique à deux sens (dans la figure 1, le concept commun le plus spécifique à s_1 et s_2 et s_3 – que l'on notera par

5. Par exemple, «haut-de-forme» est un hyponyme de «chapeau». Plus généralement, un mot A est hyponyme de B si l'extension du signifié de A est incluse dans l'extension du signifié de B.

6. Par exemple, «maison» est un holonyme de «toit». Plus généralement, un holonyme A d'un mot B est un mot dont le signifié désigne un ensemble comprenant le signifié de B.

la suite $\mathrm{lso}(s_1, s_2) = s_3$ comme *lowest superordinate* –). On calculera la quantité d'information à partir de la fréquence d'apparition du sens s_3 dans la langue : $\mathrm{IC}(s_3) = -\log(P(s_3))$. La mesure de similarité de Resnik s'exprime de la manière suivante, se servant de ces définitions :

$$\mathrm{sim}_{\mathrm{Res}} = \mathrm{IC}(\mathrm{lso}(s_1, s_2)) = \mathrm{IC}(s_3)$$

Cependant, en procédant ainsi, on perd la précision apportée par les sens s_1 et s_2 qui sont plus précis – et ainsi contiennent plus d'informations –. Jiang et Conrath [10] proposent d'utiliser la mesure suivante :

$$\mathrm{sim}_{\mathrm{JCN}} = \mathrm{IC}(s_1) + \mathrm{IC}(s_2) + 2\mathrm{IC}(\mathrm{lso}(s_1, s_2))$$

Seco *et al.*, en 2004 [22], ont suggéré qu'il était possible d'extraire directement de WordNet les valeurs de contenu informationnel, sans avoir à passer par un corpus. La taxinomie construite par WordNet étant cosntruite à partir de principes psycholinguistiques, on peut faire l'hypothèse que les liens d'hyponymie et hyperonymie sont représentatifs du contenu informationnel. En effet, si un concept possède beaucoup d'hyponymes, il est fort possible qu'il contiennent moins d'information que les concepts se trouvant en profondeur dans l'arbre taxonomique. Il définit le contenu informationnel intrinsèque en se servant de $\mathrm{hypo}(s)$, le nombre d'hyponymes du sens s, et de $\mathrm{max}_{\mathrm{wn}}$ le nombre total de concepts présents dans la taxonomie :

$$\mathrm{iIC}(s) = 1 - \frac{\log(\mathrm{hypo}(s) + 1)}{\mathrm{max}_{\mathrm{wn}}}$$

Cela permet d'éliminer le besoin d'utiliser un apprentissage non supervisé dans ces mesures. Néanmoins, cette mesure de contenu informationnel est limitée : elle n'utilise qu'un seul type de relations. Pirró et Euzenat [18] ont donc proposé de l'étendre en l'exprimant de la manière suivante :

$$\mathrm{eIC}(s) = \zeta \mathrm{iIC}(s) + \eta \mathrm{EIC}(s)$$

où ζ et η sont des constantes, $\mathrm{EIC}(s)$ la somme des iIC moyennes, des concepts reliés à s par les relations taxonomiques. En notant $\mathrm{Rels}(s)$ l'ensemble des relations possibles pour s et pour une relation $s_r \in \mathrm{Rels}(s)$, $s_r(s)$ est l'ensemble des concepts reliés à s par s_r, on a : $\mathrm{EIC}(s) = \sum_{s_r \in \mathrm{Rels}(s)} \frac{\sum_{c \in s_r(s)} \mathrm{iIC}(c)}{|s_r(s)|}$.

Le lecteur intéressé par les mesures sémantiques pourra aussi consulter l'article de Panchenko [17], qui compare 34 mesures sémantiques différentes (parmi lesquelles on retrouve celles présentées dans ce rapport) et qui constate que les mesures combinées sont les plus performantes.

I.2. DÉSAMBIGUÏSATION LEXICALE

Les mesures de similarité sont utilisées par divers algorithmes dans le but de désambiguïser des textes. On présente, dans cette section, un certain nombre de ces algorithmes.

I.2.1. ALGORITHMES GLOBAUX

La première approche adoptée par Lesk [12] pour désambiguïser un texte en entier est de construire toutes les combinaisons possibles de sens, les évaluer, et de choisir celles qui maximisent le score du texte, qui est calculé de la manière suivante : si S_w est le sens sélectionné pour le mot w dans la combinaison C, et le texte T est une liste ordonnée de mots, alors on définit le score de la combinaison par :

$$\mathrm{score}(C) = \sum_{w_i \in T} \sum_{w_j \in T} \mathrm{sim}(S_{w_i}, S_{w_j})$$

Cette méthode nécessite d'évaluer un nombre exponentiel de combinaisons ($\prod_{w \in T} N_w$ combinaisons). Pour diminuer le temps de calcul, on peut utiliser une fenêtre autour du mot, lors du calcul du score, mais on perd alors la cohérence globale de la désambiguïsation.

Hirst et St-Onge, en 1998 [9] ont proposé un algorithme global se basant sur la construction de chaînes lexicales, qui intégrant des connaissances linguistiques, pour pouvoir réduire l'espace de recherche.

On construit la chaîne lexicale globale dans l'ordre des mots du texte : si un mot est inséré (soit en étant présent dans le texte, soit via une relation), on y relie un certain nombre de ses *synsets*. Si c'est le premier mot de la chaîne, ou qu'il provient d'une relation très forte, on conserve l'ensemble de ses *synsets*. Par contre, lorsqu'il provient d'un lien fort, on n'inclut que les *synsets* qui sont reliés par des liens forts. Lorsque le mot provient d'une relation moyennement forte, on n'inclut que les *synsets* avec le meilleur score. Tous les autres *synsets* sont ignorés. On ignore aussi les *synsets* lorsqu'ils forment une chaîne illégale. Lors de l'insertion d'un mot, on cherche parmi les relations par ordre décroissant de force, si une chaîne existe déjà, dans un contexte de plus en plus petit (respectivement, généralement, toutes les phrases, sept phrases, puis trois phrases). Si une chaîne existe, le mot y est ajouté. Sinon, une nouvelle chaîne est créée.

Cette méthode est néanmoins peu précise puisqu'elle est gloutonne [15] et différentes améliorations ont été proposées, comme avec Barzilay et Elhadad [2] qui conservent toutes les interprétations possibles, augmentant la précision au détriment des performances, et Silber et McCoy [23] qui proposent un algorithme de construction de chaines lexicales linéaires, permettant d'avoir une meilleure précision sans réduire les performances.

I.2.2. Recuit simulé

Le recuit simulé, méthode d'optimisation stochastique classique, a été appliqué à la désambiguïsation lexicale par Cowie *et al.* en 1992 [4]. Cette méthode consiste à faire des changements aléatoires dans la configuration [7] de l'espace de recherche, puis d'évaluer si ce changement est bénéfique. Si c'est le cas, on conserve le changement.

L'algorithme est expliqué plus en détail dans l'annexe A.

I.2.3. Algorithme génétique

Gelbukh *et al.*, en 2003[7] ont appliqué à la désambiguïsation lexicale les algorithmes génétiques, inspirés de l'évolution génétique des espèces. On représente la configuration utilisée de la même manière que pour le recuit simulé, mais on considère désormais une population λ de configurations (assimilables à des chromosomes). Chaque indice du vecteur d'une configuration est cosidéré comme un allèle, et les allèles possibles pour un indice sont les différents sens du mot auquel l'indice fait référence.

Comme son nom l'indique, l'algorithme s'inspire des cycles reproductifs et de la sélection naturelle des espèces. La *qualité* d'un individu – qui correspond à une configuration – est estimée comme étant le score de la configuration. Lors de chaque cycle, on calcule les scores de chacun des individus et on sélectionne un nombre pair d'individus pour le croisement, où $\mathrm{score}_{\max}$ est le meileur score dans la population, et Cr un rapport de sélection :

$$\forall \lambda_i \in \lambda : \mathrm{p}_{\mathrm{crois}_{\lambda_i}} = \mathrm{Cr} \times \left(\frac{\mathrm{score}(\lambda_i)}{\mathrm{score}_{\max}} \right)$$

Le croisement des individus est alors réalisé, en effectuant une permutation autour d'un ou deux pivots choisis au hasard dans la configuration. Les individus qui ne sont pas retenus pour le croisement sont dupliqués. On obtient alors une nouvelle population, où sur chaque individu, Mn changements aléatoires uniformes seront appliqués avec une probabilité $p(M)$. Le score de la nouvelle population sera calculé après la phase de mutation, et ensuite le nouveau cycle commencera.

I.2.4. Exploration pseudo-aléatoire de graphes : PageRank

On peut aussi effectuer cette désambiguïsation en effectuant une marche aléatoire dans un graphe où l'on retrouve les différents sens possibles des mots.

L'algorithme de *PageRank* conçu par Brin et Page en 1998 [3] a été appliqué à la désambiguïsation lexicale par Mihalcea *et al.* en 2004 [13]. Dans cet algorithme, on assigne récursivement des poids aux différents arcs d'un graphe, en exploitant les informations disponibles. Dans leur algorithme, Mihalcea *et al.* utilisent WordNet et ses relations pour construire le graphe décrit dans le paragraphe précédent. Pour un mot, l'ensemble des *synsets* qui lui sont liés constituent les noeuds du graphes, et les arcs sont les relations issues de WordNet entre les *synsets* et les mots du

7. qui, ici, correspond aux numéros de sens sélectionnés pour chaque mot, en fonction de l'ordre d'apparition des mots dans le texte

texte. Les *synsets* du même mot ne sont jamais reliés entre eux (un mot n'ayant que rarement deux sens différents dans un même texte).

Ce graphe construit, on assigne un poids initial aux arcs – pouvant être soit le même pour tous les arcs, soit généré par une mesure de similarité sémantique – et on commence la marche aléatoire. Le marcheur va choisir un arc parmi ceux possibles suivant les valeurs des nœuds reliés au nœud courant. On met à jour le score d'un noeud à chaque passage sur un noeud avec la formule suivante :

$$S(N_i) = (1-d) + d \times \sum_{j \in \mathrm{In}(N_i)} \frac{S(N_j)}{|\mathrm{Out}(N_j)|}$$

avec d un facteur de lissage, N_i un noeud du graphe, $\mathrm{In}(N_i)$ l'ensemble des nœuds prédécesseurs de N_i et $\mathrm{Out}(N_j)$ l'ensemble des nœuds successeurs de N_j [8]. Une fois que le système a convergé sur une distribution stationnaire, le sens correspondant au nœud avec le meilleur score PageRank est sélectionné comme étant le sens du mot dans le texte.

I.2.5. Exploration pseudo-aléatoire de graphes : colonies de fourmis

Comme son nom l'indique, les algorithmes à colonies de fourmis cherchent à imiter le fonctionnement d'une colonie de fourmis. Cette approche, initialement proposée par Dorigo en 1992 [6], part de la constatation, faite par Deneubourg *et al.* en 1983 [5] que les fourmis vont converger systématiquement vers le chemin le plus court pour trouver de la nourriture. Lorsqu'elles empruntent un chemin, les fourmis déposent des phéromones afin d'alerter les autres fourmis de la présence de nourriture. Cette substance s'évapore naturellement, ce qui permet de facilement éliminer des chemins qui mènent à une réserve de nourriture épuisée.

Guinand et Lafourcade, en 2010 [8] ont proposé d'appliquer ce type d'algorithme à la désambiguïsation lexicale, utilisant un modèle de similarité à base de vecteurs pour régir le déplacement des fourmis. Schwab *et al.* [21] ont remplacé ce modèle par l'utilisation de la mesure de Lesk étendu. Le graphe utilisé dans ce deuxième algorithme ne relie pas les sens entre eux dans un premier temps, mais correspond à un arbre reprenant la structure de la phrase – et dont les feuilles sont les sens possibles des mots.

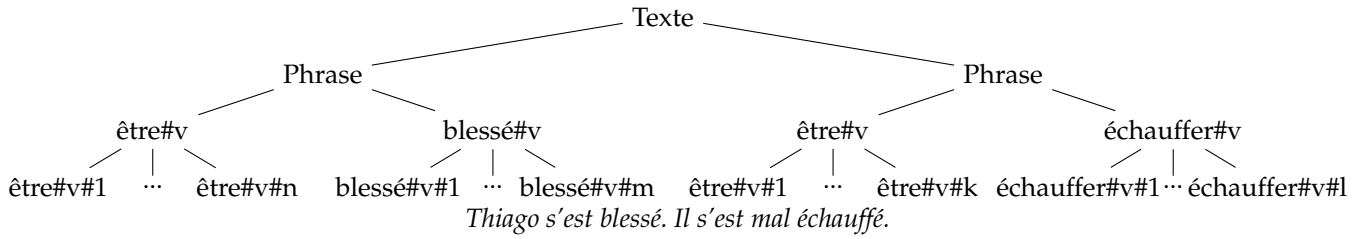

FIGURE 2 – Exemple de graphe utilisé dans l'algorithme à colonies de fourmis

La racine de l'arbre utilisé dans cette application de l'algorithme à colonies de fourmis est un nœud qui correspond au texte, les nœuds fils correspondent aux phrases, les fils suivants aux mots et enfin, les feuilles correspondent aux sens. Ces feuilles seront les fourmilières de notre graphe, et produiront à chaque cycle des fourmis. Ces fourmilières possèdent pour cela une réserve d'énergie. Au fur et à mesure des cycles, des «ponts» pourront être formés entre les feuilles par les fourmis [9]. On attache un vecteur de sens [10] à tous les nœuds qui ne sont pas des fourmilières.

Les fourmis vont partir à l'exploration de ce graphe de manière pseudo-aléatoire : ici, la probabilité de sélection d'un chemin dépend de la quantité d'énergie se trouvant sur le nœud pouvant être atteint, de la concentration de phéromone sur l'arc, et du score entre le vecteur de sens présent sur le nœud et sa fourmilière d'origine [11]. Lorsque la fourmi arrive sur le nœud, elle prélève une certaine quantité d'énergie et peut décider de revenir à sa fourmilière

8. Le graphe construit pouvant être un graphe orienté, si l'on ne prend en compte que les relations dans un sens et pas dans l'autre (l'hyponymie et la méronymie, par exemple)
9. Le graphe perdant effectivement à ce moment là sa structure d'arbre.
10. Ce vecteur contenant les identifiants des sens utilisés par WordNet
11. On utilise pour le calcul du score des mesures de similarité locale – en particulier Lesk étendu

suivant une probabilité dépendant de l'énergie qu'elle transporte. Lorsque la fourmi passe sur un nœud qui n'est pas une fourmilière, elle dépose le sens correspondant à sa fourmilière dans le vecteur de sens du nœud, ainsi qu'une certaine quantité de phéromone, qui s'évaporera partiellement à chaque cycle.

Si une fourmi arrive sur une fourmilière qui porte un sens différent du sien, elle peut construire un pont vers sa fourmilière pour y revenir directement, suivant une certaine probabilité, dépendant du score entre les deux sens. Ce pont se comporte alors comme n'importe quel arc. Lorsque de nombreux ponts ont été construits, certains se renforceront et d'autres disparaîtront (avec l'évaporation des phéromones), menant progressivement à une monopolisation des ressources entre les fourmilières aux ponts les plus fréquentés. On obtient ainsi avec ces ponts des chemins interprétatifs, parmi toutes les combinaisons de sens possibles. À la fin de la simulation, les sens correspondants aux fourmilières avec le plus d'énergie sont choisis.

II — Analyse des paramètres d'un algorithme à colonies de fourmis

L'algorithme à colonies de fourmis présenté dans la section précédente doit toujours être optimisé. Pour cela, il faut jouer sur un certain nombre de paramètres : le choix de la mesure de similarité (dans ce stage, la seule mesure de similarité utilisée est Lesk Étendu), et le choix des colonies de fourmis formées (qui dépend de la taille du texte, et qui dépend aussi du temps d'exécution traité). Mais ce ne sont pas les seuls paramètres à prendre en compte : dans cette section, en utilisant Lesk Étendu et le texte 1 de la tâche 7 de Semeval 2007 [16], nous allons nous intéresser à l'ajustement des paramètres, en les analysant.

II.1. Paramètres de l'algorithme

Cet algorithme utilise plusieurs paramètres différents, définis avant l'exécution de l'algorithme :
- `AntLife`, le nombre de cycles pendant lesquels la fourmi parcourt le graphe avant de mourir et disparaître ;
- `Cycles` le nombre de cycles que l'algorithme effectuera ;
- `GetEnergy` l'énergie que la fourmi récupère lors de son passage sur un nœud ;
- `InitialEnergy` l'énergie initiale des fourmilières ;
- `MaxEnergy` l'énergie maximale que peut transporter une fourmi ;
- `phEvap` le taux d'évaporation de phéromone (en pourcentage, exprimé entre 0 et 1).
- `TailleVect` la taille du vecteur de sens présent sur les nœuds
- et `DepotVect` le taux de dépôt du vecteur de sens de la fourmi (et de sa fourmilière) lors de son passage sur un nœud.

Certains paramètres ont été éliminés avant d'effectuer l'analyse : les concepteurs de l'algorithme ont en effet constaté empiriquement que l'algorithme convergeait au delà d'environ 80 cycles. Pour conserver une marge d'erreur et pouvoir effectivement avoir la convergence de l'algorithme à chacun de nos essais, le nombre de cycles effectués par l'algorithme à colonies de fourmis a été fixé à 100.

Paramètre	Intervalle	Valeur initiale	Écart choisi
`AntLife`	1 – 30	10	$\Delta = 1$
`InitialEnergy`	1 – 30	5	$\Delta = 1$
`GetEnergy`	1 – 20	10	$\Delta = 1$
`MaxEnergy`	1 – 60	30	$\Delta = 2$
`phEvap`	0.02 – 1	0.3	$\Delta = 0.02$
`TailleVect`	20 – 200	100	$\Delta = 5$
`DepotVect`	0.02 – 1	0.4	$\Delta = 0.02$

FIGURE 3 – Intervalles choisis de valeurs pour les paramètres

Les valeurs présentées dans la figure 3 ont été choisies empiriquement en fonction des essais réalisés sur l'algorithme sur la campagne d'évaluation Semeval 2007 [16] pendant sa conception. Elles pourront être modifiées si l'analyse des paramètres montre que la valeur d'un paramètre en particulier doit être choisie comme plus grande ou plus petite.

II.2. MÉTHODE D'ANALYSE

Pour effectuer cette analyse de paramètres, qui consiste en fait à la recherche de la meilleure combinaison de paramètres possible, nous allons utiliser un algorithme de recuit simulé (qui a été présenté en section 1.2.2).

II.2.1. RECUIT SIMULÉ UTILISÉ

Un certain nombre de paramètres doivent être définis pour faire fonctionner l'algorithme du recuit simulé, comme présenté dans la section 1.2.2 :

- la loi de température choisie est classique, mais avec un facteur de diminution plus important : $T_n = 0,8^n T_0$. Cela permet à l'algorithme de converger plus rapidement, même s'il n'atteindra pas forcément le maximum possible pour le texte choisi [12].
- la conservation ou non de la nouvelle configuration de paramètres, après 100 exécutions de l'algorithme à colonies de fourmis, dépend de plusieurs critères :
  - la valeur p (*p-value*) [13] entre les résultats de l'ancienne et de la nouvelle configuration doit être en dessous de $0,05$, afin de montrer que le changement de paramètre effectué a réellement une importance. Si cette valeur p est au dessus de $0,05$, la différence entre les résultats des algorithmes – qui restent aléatoires puisqu'on se base sur une marche aléatoire – ne pourra pas être considérée comme due au changement de paramètre effectué
  - Si cette valeur p est en dessous du seuil, la configuration est toujours conservée si elle améliore le meilleur score. Si ce n'est pas le cas, on la conserve avec la probabilité suivante :

$$p_{\text{conservation}} = e^{100 \times \frac{s_{\text{nouvelleconfig}} - s_{\text{ancienneconfiguration}}}{T}}$$

    où s_{conf} est le score moyen obtenu par l'algorithme avec la configuration *conf*. Celui-ci est, pour une exécution donnée de l'algorithme à colonies de fourmis utilisé, égal à $\frac{\text{sens correctement devinés}}{\text{sens à choisir}}$. Le choix de multiplier cette valeur par 100 permet d'éviter que les changements soient trop rares.

L'algorithme se terminera lorsque les deux conditions suivantes seront remplies : 21 cycles doivent être effectués au minimum, et la configuration conservée au $n+1^{\text{ème}}$ cycle devra être différente de la configuration conservée au $n^{\text{ème}}$ cycle.

II.2.2. CHOIX DE CONFIGURATION ET CORPUS

Lors de chaque début de cycle de cet algorithme de recherche, une modification de la configuration est effectuée : on part de la configuration conservée, on choisit au hasard l'un des six paramètres, et on choisit – de nouveau au hasard – une nouvelle valeur de ce paramètre dans l'espace défini dans la figure 3. Ce choix de modification permet d'analyser l'importance de la modification d'un seul paramètre, et ainsi de peut-être réduire l'espace de recherche si seulement quelques paramètres sont réellement significatifs. Il est aussi laissé la possibilité à l'algorithme de n'effectuer aucun changement [14]

L'algorithme à colonies de fourmis sera, dans cette analyse de paramètres, seulement executé sur le texte 1 de la tâche 7 de Semeval 2007 [16], afin de réduire le temps d'exécution de l'algorithme à un temps raisonnable pour pouvoir effectuer 100 exécutions par cycle de notre algorithme de recherche. [15].

12. Cela sera discuté de nouveau plus loin dans ce rapport
13. La valeur p, comprise entre 0 et 1, se calcule entre deux distributions de résultats, afin de déterminer si il est effectivement possible ou non que ces distributions proviennent d'un tirage aléatoire effectué avec la même méthode.
14. Ceci permettant de s'assurer un peu plus de la validité de la méthode
15. Ce choix sera aussi discuté plus loin dans le rapport

II.3. RÉSULTATS OBTENUS

II.3.1. ANALYSE DES PARAMÈTRES MODIFIANT LE COMPORTEMENT DE L'ALGORITHME

Paramètre modifié	Valeur $p < 0.05$	$0.05 \leq$ Valeur $p < 0.1$	Valeur $p \geq 0.1$
`AntLife`	15	0	2
`InitialEnergy`	18	1	1
`MaxEnergy`	9	0	0
`GetEnergy`	7	0	4
`phEvap`	4	1	12
`TailleVect`	2	1	11
`DepotVect`	0	0	13
Pas de modification	1	0	5

FIGURE 4 – Influence des changements de paramètres

Même si le résultat de l'algorithme dépend de l'ensemble des paramètres choisis à son exécution, lorsqu'on s'intéresse aux modifications d'un seul paramètre (entre deux cycles de l'exécution de notre algorithme de recherche), on peut déjà effectuer un premier tri dans l'ensemble des paramètres à tester :

- L'apparition d'`AntLife` parmi les paramètres importants était attendu, puisque ce paramètre règle la durée de vie de la fourmi sur l'algorithme : s'il n'est pas suffisemment long, la fourmi va mourir avant de trouver assez d'énergie, et ne pourra pas effectuer un parcours suffisant pour donner une information sur le sens des mots. Au contraire, si la fourmi vit suffisemment longtemps, elle pourra revenir à une fourmilière avant de mourir, et ainsi contribuer à la formation des ponts entre fourmilières.
- les paramètres liés à l'énergie (`InitialEnergy`, `GetEnergy` et `MaxEnergy`) semblent importants dans l'exécution de l'algorithme à colonies de fourmis.
- les paramètres concernant les vecteurs de sens apposés sur les nœuds ne semblent, quant à eux, pas prépondérants dans l'exécution de l'algorithme.

L'absence de modification des paramètres, comme on pouvait s'y attendre, n'entraîne pas de réelle modification dans le fonctionnement de l'algorithme. Ce n'était en réalité pas le cas si l'algorithme n'était exécuté seulement 30 fois, et a donné lieu au choix de l'exécuter 100 fois avant de le comparer aux exécutions précédentes, afin que les distributions de scores obtenus par les 100 exécutions soient suffisemment représentatives.

II.3.2. CHOIX DES VALEURS DES PARAMÈTRES POUR AMÉLIORER LE SCORE OBTENU

Maintenant que les paramètres pouvant effectivement influencer sur le résultat de l'algorithme ont été identifiés, il faut effectuer une analyse plus précise, portant sur les valeurs des paramètres. Il y a deux questions à résoudre :

- Dans quels intervalles de valeurs d'un paramètre observe-t-on des modifications importantes ?
- Quelles sont les valeurs de paramètres donnant un meilleur résultat ?

On effectue toujours un recuit simulé pour essayer d'aboutir aux meilleures configurations plus rapidement et on s'intéressera aux modifications importantes (celles pour lesquelles la valeur p est inférieure à 0,05) et au fait que ces modifications améliorent ou non le score de l'algorithme sur le corpus.

La durée du stage n'a malheureusement pas été assez importante pour pouvoir obtenir un échantillon suffisemment représentatif de résultats pour pouvoir les présenter et les analyser.

Il aurait ensuite fallu regarder si ces modifications de paramètres avaient une influence globale ou non, en récupérant les meilleures valeurs obtenus pour le premier corpus et en les utilisant sur un second corpus, suffisemment de fois pour avoir un score moyen représentatif – et ensuite refaire la procédure pour voir si les meilleures valeurs pour le nouveau corpus sont suffisemment proches des valeurs de paramètres que l'on avait trouvé.

CONCLUSION

Nous avons donc ici étudié diverses modifications des paramètres pour pouvoir, dans un premier temps, avoir une idée précise des paramètres influençant effectivement le fonctionnement de l'algorithme. Une fois les paramètres qui n'ont pas d'importance dans le fonctionnement de l'algorithme à colonies de fourmis, nous avons ensuite utilisé le principe du recuit simulé pour trouver les meilleurs paramètres possibles parmi l'espace de paramètres issu des expériences manuelles précédentes.

Néanmoins, nous n'avons pas eu le temps de tester l'influence des paramètres sur un autre texte – ni d'affiner l'espace de recherche – ce qui nous ne permet pas d'affirmer que ces paramètres seront effectivement les meilleurs pour l'ensemble des textes que l'on souhaiterait analyser.

Ce stage m'a permis d'appliquer certaines méthodes utilisées dans le domaine de l'apprentissage, et de les approfondir, tout en améliorant mes connaissances sur le thème – qui attirait ma curiosité – du traitement automatique des langues. J'ai ainsi passé six semaines agréables au sein d'une équipe motivante et très intéressée par le traitement automatique des langues.

REMERCIEMENTS

Je tiens à remercier Didier SCHWAB et Jérôme GOULIAN de m'avoir proposé un sujet et accepté en tant que stagiaire. Je tiens aussi à les remercier pour l'invitation aux journées JEP-TALN-RECITAL organisées cette année à Grenoble, ce qui m'a permis de découvrir encore plus précisément l'étendue du domaine du traitement des langues. Je les remercie de nouveau, cette fois-ci avec un autre stagiaire, Andon TCHECHMEDJIEV – que je félicite par ailleurs pour avoir obtenu sa bourse de thèse – pour l'ensemble des explications sur leurs travaux et leurs clarifications sur les choses que je n'avais pas comprises dans un premier temps.

ANNEXE A — RECUIT SIMULÉ

Le recuit simulé est une méthode empirique d'optimisation, inspirée d'un processus utilisé en métallurgie où l'on alterne des cycles de refroidissement et de réchauffage (d'où le nom de recuit) afin de minimiser l'énergie du matériau. On cherche ici a trouver les extrêma d'une fonction. Par analogie à la métallurgie, elle repose donc sur deux paramètres : une température T, et une mesure initiale E.

Pour l'initialisation, on choisit une solution possible au hasard dans l'espace des solutions possibles. On calcule son énergie $E = E_0$ en fonction des critères que l'on cherche à maximiser ou diminuer. Pour conserver l'analogie, on considèrera fonction d'énergie choisie décroissante par rapport à l'amélioration des solutions. On choisit aussi une température $T = T_0$, généralement élevée, dépendant de la loi de décroissance qui sera utilisée.

L'algorithme fonctionnera pendant un certain nombre d'itérations (ou de cycles) pendant lesquels :
- on effectue dans un premier temps une modification simple de la solution choisie jusqu'alors.
- on calcule l'énergie de cette nouvelle solution (qu'on notera E') et on la compare à l'énergie de la solution conservée (notée E') :
  - Si $E' < E$, on minimise l'énergie (et on améliore donc la solution) : on va garder la modification effectuée, et la nouvelle solution proposée sera conservée.
  - Sinon, on accepte cette solution avec une probabilité $e^{-\frac{\Delta E}{T}}$ [16].

L'évolution de la température est gérée de deux façons différentes dans les recuits simulés :
- Soit l'itération se fait à température constante. Lorsque l'énergie demeure constante au bout d'un certain nombre de modifications apportées à la solution, on abaisse la température du système. On fonctionne alors par *paliers* de température.
- Soit on fait évoluer la température de manière continue, le plus souvent avec $T_{n+1} = \lambda T$ où $0 < \lambda \leq 1$ (où $\lambda = 0,99$ le plus couramment).

L'algorithme s'arrête lorsque la température a baissé au delà d'un certain seuil ou lorsque l'énergie reste figée pendant assez longtemps.

On remarque que la température joue un rôle important dans cet algorithme : à haute température, la probabilité d'accepter une solution – même très mauvaise – est très élevée. À plus basse température, il reste possible d'aller regarder des solutions apparemment mauvaises, à partir desquelles les modifications simples pourront en réalité permettre de s'approcher de l'extremum souhaité global alors que l'on aurait été piégé dans un extremum local sinon.

16. Cette probabilité est la règle de Metropolis

RÉFÉRENCES

[1] S. Banerjee and T. Pedersen. An adapted lesk algorithm for word sens disambiguation using wordnet. In *CICLin '02*, pages 136–145, London, UK, 2002.

[2] R. Barzilay and M. Elhadad. Using lexical chains for text summarization. In *Proceedings of the ACL Workshop on Intelligent Scalable Text Summarization*, pages 10–17, 1997.

[3] S. Brin and L. Page. The anatomy of a large-scale hypertextual web search engine. In *WWW7*, pages 107–117, Amsterdam, 1998.

[4] J. Cowie, J. Guthrie, and L. Guthrie. Lexical disambiguation using simulated annealing. In *COLING '92*, pages 359–365, Stroudsburg, PA, USA, 1992. ACL.

[5] Pasteels J. M. Deneubourg, J. L. and J. C. Verhaege. Probabilistic behaviour in ants : a strategy of errors ? *Journal of Theoretical Biology*, 105 :259–271, 1983.

[6] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, 1992.

[7] A. Gelbukh, G. Sidorov, and S.-Y. Han. Evolutionary approach to natural language word sense disambiguation through global coherence optimization. *WSEAS Transactions on Communications*, 1(2) :11–19, 2003.

[8] F. Guinand and M. Lafourcade. *Artifical ants for Natural Language Processing*, chapter 20, pages 455–492. 2010.

[9] G. Hirst and D. D. St-Onge. *Lexical chains as representations of context for the detection and correction of malapropisms*, pages 305–332. MIT Press, Cambridge, MA, 1998.

[10] J. J. Jiang and D. W. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In *ROCLING X*, 1997.

[11] C. Leacock and M. Chodorow. *WordNet : An Electronic Lexical Database*. MIT Press, Cambridge, MA, 1998.

[12] M. Lesk. Automatic sense disambiguation using machine readable dictionaries : how to tell a pine cone from an ice cream cone. In *SIGDOC '86*, New York, NY, USA. ACM.

[13] R. Mihalcea, P. Tarau, and E. Figa. Pagerank on semantic networks, with application to word sense disambiguation. In *COLING '04*, Stroudsburg, PA, USA, 2004. ACL.

[14] G.A. Miller. Wordnet : a lexical database for english. *Communications of the ACM*, 38, 1995.

[15] R. Navigli. Word sens disambiguation : A survey. *ACM Computing Surveys*, 41(2) :1–10, 2009.

[16] R. Navigli, K. Litkowski, and O. Hargraves. Semeval-2007 task 07 : Coarse-grained english all-words task. In *Proceedings of Semeval-2007 Workshop (SEMEVAL), 45th Annual Meeting of the Association for Computational Linguistics*, Prague, Czech Republic. ACL.

[17] A. Panchenko. A study of heterogeneous similarity measures for semantic relation extraction. volume 3 : RECITAL, pages 29–42, Grenoble, 2012.

[18] G. Pirró and J. Euzenat. A feature and information theoretic framework for smeantic similarity and relatedness. *Lecture Notes in Computer Science*, 6946 :615–630, 2010.

[19] R. Rada, H Mili, E. Bicknell, and M. Blettner. Development and application of a metric on semantic nets. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(1) :17–30, 1989.

[20] P. Resnik. Using information content to evaluate semantic similarity in a taxonomy. In *IJCAI'95*, pages 448–453, San Francisco, CA, USA, 1995.

[21] D. Schwab, J. Goulian, and N. Guillaume. Désambiguïsation lexicale par propagation de mesures sémantiques locales par algorithmes à colonies de fourmis. In *TALN*, Montpellier.

[22] N. Seco, T. Veale, and J. Hayes. An intrinsic information content metric for semantic similarity in wordnet. In *Proceedings of ECAI '2004*, pages 1089–1090, Valencia, Spain, 2004.

[23] H. G. Silber and K. F. McCoy. Efficient text summarization using lexical chains. In *IUI '00*, pages 252–55, New York, NY, USA, 2000. ACM.

[24] A.M. Turing. Computing machinery and intelligence. *Mind*, LIX(236) :433–460, 1950.

[25] A. Tversky. Features of simiarity. *Psychological Review*, 84(4) :327–352, 1977.

[26] Y. Wilks and M. Stevenson. Word sense disambiguation using optimised combinations of knowledge sources. In *COLING '98*, pages 1398–1402, Stroudsburg, PA, USA, 1998. ACL.

[27] Z. Wu and M. Palmer. Verbs semantics and lexical selection. In *Proceedings of the 32nd annual meeting on ACL*, volume 2 de *ACL '94*, pages 133–138, Stroudsburg, PA, USA, 1994. ACL.

[28] G.K. Zipf. *Human Behavior and the Principle of Least Effort*. Addison-Wesley, 1949.