TALN 2011, Montpellier, 27 *juin* –1er *juillet 2011*

Désambiguïsation lexicale par propagation de mesures sémantiques locales par algorithmes à colonies de fourmis

Didier Schwab, Jérôme Goulian, Nathan Guillaume
LIG-GETALP (Laboratoire d'Informatique de Grenoble, Groupe d'Étude pour la Traduction/le Traitement Automatique des Langues et de la Parole)
Université Pierre Mendès France, Grenoble 2
{didier.schwab, jerome.goulian}@imag.fr

Résumé. Effectuer une tâche de désambiguïsation lexicale peut permettre d'améliorer de nombreuses applications du traitement automatique des langues comme l'extraction d'informations multilingues, ou la traduction automatique. Schématiquement, il s'agit de choisir quel est le sens le plus approprié pour chaque mot d'un texte. Une des approches classiques consiste à estimer la proximité sémantique qui existe entre deux sens de mots puis de l'étendre à l'ensemble du texte. La méthode la plus directe donne un score à toutes les paires de sens de mots puis choisit la chaîne de sens qui a le meilleur score. La complexité de cet algorithme est exponentielle et le contexte qu'il est calculatoirement possible d'utiliser s'en trouve réduit. Il ne s'agit donc pas d'une solution viable. Dans cet article, nous nous intéressons à une autre méthode, l'adaptation d'un algorithme à colonies de fourmis. Nous présentons ses caractéristiques et montrons qu'il permet de propager à un niveau global les résultats des algorithmes locaux et de tenir compte d'un contexte plus long et plus approprié en un temps raisonnable.

Abstract. Word sense disambiguation can lead to significant improvement in many Natural Language Processing applications as Machine Translation or Multilingual Information Retrieval. Basically, the aim is to choose for each word in a text its best sense. One of the most popular method estimates local semantic relatedness between two word senses and then extends it to the whole text. The most direct method computes a rough score for every pair of word senses and chooses the lexical chain that has the best score. The complexity of this algorithm is exponential and the context that it is computationally possible to use is reduced. Brute force is therefore not a viable solution. In this paper, we focus on another method : the adaptation of an ant colony algorithm. We present its features and show that it can spread at a global level the results of local algorithms and consider a longer and more appropriate context in a reasonable time.

Mots-clés : Désambiguïsation lexicale, Algorithmes à colonies de fourmis, Mesures sémantiques.

Keywords: Lexical Disambiguation, Ant colony algorithms, Semantic relatedness.

1 Introduction

Effectuer une tâche de désambiguïsation lexicale peut permettre d'améliorer de nombreuses applications du traitement automatique des langues comme l'extraction d'informations multilingues, le résumé automatique ou encore la traduction automatique. Schématiquement, il s'agit de choisir quel est le sens le plus approprié pour chaque mot d'un texte dans un inventaire pré-défini. Par exemple, dans "*La souris mange le fromage.*", l'animal devrait être

préféré au dispositif électronique. De nombreux travaux existent sur le sujet, que l'on sépare habituellement en approches supervisées et non-supervisées. Les premières utilisent des apprentissages réalisés grâce à des corpus manuellement annotés, les secondes n'utilisent pas de telles données. Une catégorie intermédiaire, constituée des approches semi-supervisées, utilise quelques données annotées comme, par exemple, un sens par défaut issu d'un corpus annoté lorsque l'algorithme principal échoue (Navigli & Lapata, 2010). Le lecteur pourra consulter (Ide & Véronis, 1998) pour les travaux antérieurs à 1998 et (Agirre & Edmonds, 2006) ou (Navigli, 2009) pour un état de l'art complet.

La création de données annotées est une opération compliquée puisqu'elle nécessite une importante main d'œuvre et qu'elle doit être réalisée pour chaque inventaire de sens, pour chaque langue et même pour chaque domaine spécifique (sport, finance, ...). Cette constatation, que nous partageons avec (Navigli & Lapata, 2010), nous conduit à nous intéresser plus particulièrement à des approches non-supervisées. Une de ces approches classiques consiste à estimer la proximité sémantique qui existe entre deux sens de mots puis de l'étendre à l'ensemble du texte. En d'autres termes, il s'agit de donner des scores locaux et de les propager au niveau global (phrase, paragraphe, texte, ...). La méthode la plus directe, utilisée par exemple par (Pedersen *et al.*, 2005) utilise un algorithme brutal qui donne un score à toutes les paires de sens de mots puis choisit la chaîne de sens qui a le meilleur score. La complexité de cet algorithme est exponentielle et le contexte qu'il est calculatoirement possible d'utiliser s'en trouve réduit. Ainsi, alors qu'une analyse au niveau de la phrase n'est déjà pas toujours possible, un contexte linguistiquement plus pertinent comme, par exemple, le paragraphe l'est encore moins.

Les applications que nous visons doivent pouvoir être utilisées en temps réel. Lorsque l'on recherche une image et encore plus lorsque l'on appelle quelqu'un qui parle une autre langue au téléphone, les réponses doivent être immédiates. Il ne s'agit donc pas une solution viable et nous étudions d'autres méthodes.

Dans cet article, nous nous intéressons à la propagation de mesures de proximité sémantique locales grâce à une adaptation d'un algorithme à colonies de fourmis. Nous présentons dans un premier temps les mesures locales que nous utilisons puis quelques unes des caractéristiques de notre algorithme de propagation. Enfin, à titre d'exemple, nous évaluons notre approche sur la tâche *gros grain* de la campagne d'évaluation Semeval 2007 (Navigli *et al.*, 2007). Nous comparons en particulier notre algorithme de propagation à l'algorithme exhaustif classique et montrons qu'il permet d'obtenir efficacement une meilleure F-mesure.

2 Algorithmes locaux

2.1 Mesures de proximité sémantique

Ces méthodes consistent à donner un score censé refléter la proximité des objets linguistiques (généralement des mots ou des sens de mots) comparés. Ces scores peuvent être des similarités, donc avoir une valeur entre 0 et 1, des distances, et donc respecter leurs trois propriétés (séparation, symétrie et inégalité triangulaire) ou plus généralement, être une valeur positive non bornée.

Parmi elles, on peut citer *Hirst & Saint-Honge* basée sur la distance en terme de graphe entre deux sens dans un réseau lexical ; *Rada et al.* ainsi que *Leacock and Chodorow* similaires à la précédente mais ne considérant que les liens de type hyperonymie ; les mesures ou distances entre vecteurs (LSA (Deerwester *et al.*, 1990), vecteurs conceptuels (Schwab, 2005)). On pourra consulter (Pedersen *et al.*, 2005), (Cramer *et al.*, 2010) ou (Navigli, 2009) pour un panorama plus complet.

En désambiguïsation lexicale, ces méthodes sont utilisées de façon locale entre deux sens de mots, et sont ensuite

appliquées à un niveau global. Dans cet article, nous nous concentrons sur l'algorithme global et, à des fins de comparaison, nous présentons deux algorithmes locaux basés sur l'algorithme de Lesk.

2.2 Algorithmes locaux de cette expérience

2.2.1 Des algorithmes inspirés par Lesk

Nous utilisons dans cet article deux variantes de l'algorithme de Lesk (Lesk, 1986). Proposées il y a plus de 25 ans, il se caractérise par sa simplicité. Il ne nécessite qu'un dictionnaire et aucun apprentissage. Le score donné à une paire de sens est le nombre de mots – ici simplement les suites de caractères séparées par des espaces – en commun dans leur définition, sans tenir compte ni de leur ordre, ni de sous-séquences communes (approche sac de mots), ni d'informations morphologiques ou syntaxiques. Les variantes de cet algorithme sont encore aujourd'hui parmi les meilleures sur l'anglais (Ponzetto & Navigli, 2010). Ce premier algorithme local est nommé dans la suite *Lesk*.

Nous utilisons WordNet (Fellbaum, 1998), une base lexicale pour l'anglais, dans laquelle les sens de mots (les synsets) sont reliés par des relations (hyperonymie, hyponymie, antonymie, *etc.*). Notre second algorithme local exploite ces liens. Au lieu d'utiliser uniquement la définition d'un sens, elle utilise également la définition des différents sens qui lui sont liés. Cette idée est similaire à celle de (Banerjee & Pedersen, 2002) [1]. Ce second algorithme local est nommé dans la suite *Lesk étendu*.

2.2.2 Efficacité algorithmique

L'algorithme de base pour comparer le nombre de mots communs à deux définitions a une complexité en $O(n \times m)$ avec n et m, les longueurs en mots des définitions. De plus, la comparaison de chaînes de caractères est une opération relativement chère. On pourrait penser qu'il suffirait de précalculer la matrice de similarités avec l'ensemble des définitions. Cette idée est utopique vu la taille que peuvent atteindre les dictionnaires (jusqu'à plusieurs millions de définitions) [2] mais aussi parce qu'on a toujours besoin de faire des calculs sur de nouvelles données puisque (1) les données et les sens peuvent évoluer au cours du temps comme dans (Schwab, 2005), (2) notre algorithme de propagation utilise des pseudo-définitions crées à la volée (voir partie 4.2.2).

Nous avons amélioré ce calcul en utilisant un prétraitement qui se déroule en deux étapes. Dans la première, nous affectons à chacun des mots trouvés dans le dictionnaire un nombre entier tandis que, dans la seconde, nous convertissons chacune des définitions en un vecteur de nombres correspondant aux mots qu'elle contient, triés du plus petit au plus grand. Nous appelons ces vecteurs, *vecteurs de définitions*.

Par exemple, si notre première étape a donné ‹‹kind››$= 1$; ‹‹of››$= 2$; ‹‹evergreen››$= 3$; ‹‹tree››$= 4$; ‹‹with››$= 5$ ‹‹needle-shaped››$= 6$; ‹‹leaves››$= 7$; ‹‹fruit››$= 8$; ‹‹certain››$= 9$ avec la définition A, "*kind of evergreen tree with needle-shaped leaves*", nous obtenons le vecteur $[1, 2, 3, 4, 5, 6, 7]$ et avec B, "*fruit of certain evergreen tree*", nous obtenons $[2, 3, 4, 8, 9]$.

Cette conversion a deux avantages : (1) la comparaison de nombres est bien plus efficace que la comparaison de chaînes de caractères, (2) ordonner ces nombres permet d'éviter des comparaisons inutiles et de gagner en

1. (Banerjee & Pedersen, 2002) introduit également une notion de sous-séquence identique dans les définitions. Nous n'avons pas encore testé cette variante dont la complexité algorithmique est nettement supérieure à celle de notre algorithme.
2. Une forme de cache pourrait en partie régler ce problème.

efficacité. Ainsi, avec ce prétraitement, la complexité passe de $O(n \times m)$ à $O(n)$ où n et m ($n \geq m$) sont les longueurs (en nombre de mots) des définitions.

Pour les définitions A et B, calculer cette même proximité sémantique avec l'algorithme sur les définitions brutes se fait en $7 \times 5 = 35$ opérations (qui plus est sur des chaînes de caractères) tandis que si les définitions sont converties en vecteurs, nous n'avons plus que 7 opérations.

3 Algorithmes globaux

L'algorithme global est l'algorithme qui va permettre de propager les résultats d'un ou plusieurs algorithmes locaux à l'ensemble du texte afin de pouvoir en déduire un sens pour chaque mot. La méthode la plus directe est la recherche exhaustive utilisée par exemple dans (Banerjee & Pedersen, 2002). Il s'agit de considérer les combinaisons de l'ensemble des sens des mots dans le même contexte (fenêtre de mots, phrase, texte, *etc.*), de donner un score à chacune de ces combinaisons et de choisir celle qui a le meilleur score. Le principal problème de cette méthode est la rapide explosion combinatoire qu'elle engendre. Considérons la phrase suivante tirée du corpus d'évaluation que nous utilisons dans la partie 5, "*The pictures they painted were flat, not round as a figure should be, and very often the feet did not look as if they were standing on the ground at all, but pointed downwards as if they were hanging in the air.*", ‹*picture*› a 9 sens, ‹*paint*› 4, ‹*be*› 13, ‹*flat*› 17, ‹*figure*› 13, ‹*very*› 2, ‹*often*› 2, ‹*foot*› 11, ‹*look*› 10, ‹*stand*› 12, ‹*ground*› 11, ‹*at all*› 1, ‹*point*› 13, ‹*downwards*› 1, ‹*hang*› 15 et ‹*air*› 9 sens, il y a alors 137 051 946 345 600 combinaisons de sens possibles à analyser. Ce nombre est comparable à la quantité d'opérations (et le calcul d'une combinaison nécessite des dizaines voire des centaines d'opérations) que peuvent théoriquement effectuer 3300 processeurs Core i7-990X (2,43GHz, 6 cœurs, 12 fils d'exécutions) sortis par Intel au premier trimestre 2011 en une seconde. Le calcul exhaustif est donc très compliqué à réaliser dans des conditions réelles et, surtout, rend impossible l'utilisation d'un contexte d'analyse plus important.

Pour contourner ce problème, plusieurs solutions ont été proposées. Par exemple, des approches utilisant un corpus pour diminuer le nombre de combinaisons à examiner comme la recherche des chaînes lexicales compatibles (Gale *et al.*, 1992; Vasilescu *et al.*, 2004) ou encore des approches issues de l'intelligence artificielle comme le recuit simulé (Cowie *et al.*, 1992) ou les algorithmes génétiques (Gelbukh *et al.*, 2003).

Ces méthodes ont en commun de ne pas permettre l'exploitation de façon directe et simple d'une structure linguistique sous forme de graphe que ce soit une analyse morphologique ou une analyse syntaxique. Nous utilisons, au contraire, une méthode à colonies de fourmis pour l'analyse sémantique inspirée de (Schwab & Lafourcade, 2007) afin de pouvoir à terme utiliser de telles structures [3].

4 Notre algorithme global : un algorithme à colonies de fourmis

4.1 Les algorithmes à colonies de fourmis

Les algorithmes à fourmis ont pour origine la biologie et les observations réalisées sur le comportement social des fourmis. En effet, ces insectes ont collectivement la capacité de trouver le plus court chemin entre leur fourmilière et une source d'énergie. Il a pu être démontré que la coopération au sein de la colonie est auto-organisée et résulte d'interactions entre individus autonomes. Ces interactions, souvent très simples, permettent à la colonie

3. Dans un premier temps, nous utiliserons ici une structure linguistique extrêmement simpl(ist)e.

de résoudre des problèmes compliqués. Ce phénomène est appelé intelligence en essaim (Bonabeau & Théraulaz, 2000). Il est de plus en plus utilisé en informatique où des systèmes de contrôle centralisés gagnent souvent à être remplacés par d'autres, fondés sur les interactions d'éléments simples.

En 1989, Jean-Louis Deneubourg étudie le comportement des fourmis biologiques dans le but de comprendre la méthode avec laquelle elles choisissent le plus court chemin et le retrouvent en cas d'obstacle. Il élabore ainsi le modèle stochastique dit *de Deneubourg* (Deneubourg *et al.*, 1989), conforme à ce qui est observé statistiquement sur les fourmis réelles quant à leur partage entre les chemins. Ce modèle stochastique est à l'origine des travaux sur les algorithmes à fourmis.

Le concept principal de l'intelligence en essaim est la stygmergie, c.-à-d. l'interaction entre agents par modification de l'environnement. Une des premières méthodes que l'on peut apparenter aux algorithmes à fourmis est l'écorésolution qui a montré la puissance d'une heuristique de résolution collective basée sur la perception locale, évitant tout parcours explicite de graphe d'états (Drogoul, 1993).

En 1992, Marco Dorigo et Luca Maria Gambardella conçoivent le premier algorithme basé sur ce paradigme pour le célèbre problème combinatoire du voyageur de commerce (Dorigo & Gambardella, 1997). Dans les algorithmes à base de fourmis artificielles, l'environnement est généralement représenté par un graphe et les fourmis virtuelles utilisent l'information accumulée sous la forme de chemins de phéromone déposée sur les arcs du graphe. De façon simple, une fourmi se contente de suivre les traces de phéromones déposées précédemment ou explore au hasard dans le but de trouver un chemin optimal, fonction du problème posé, dans le graphe.

Ces algorithmes offrent une bonne alternative à tout type de résolution de problèmes modélisables sous forme d'un graphe. Ils permettent un parcours rapide et efficace et offrent des résultats comparables à ceux obtenus par les différentes méthodes de résolution. Leur grand intérêt réside dans leur capacité à s'adapter à un changement de l'environnement. Le lecteur trouvera dans (Dorigo & Stützle, 2004) ou (Monmarche *et al.*, 2009) de bons états de l'art sur la question.

4.2 Algorithme à colonies de fourmis et désambiguïsation lexicale

4.2.1 Vue d'ensemble

L'environnement des fourmis est un graphe. Il peut être linguistique – morphologique comme dans (Rouquet *et al.*, 2010) ou morpho-syntaxique comme dans (Schwab & Lafourcade, 2007; Guinand & Lafourcade, 2009) – ou être simplement organisé en fonction des éléments du texte. En fonction de l'environnement choisi, les résultats de l'algorithme ne sont évidemment pas les mêmes. Des recherches sont actuellement menées à ce sujet mais, dans cet article, nous ne nous intéressons qu'à un cas de base c.-à-d. un graphe simple (voir fig.1), sans information linguistique externe, afin de mieux comprendre la mécanique de nos algorithmes.

Dans ce graphe, nous distinguons deux types de nœuds : les *fourmilières* et les *nœuds normaux*. Suivant les idées développées dans (Schwab, 2005) et (Guinand & Lafourcade, 2009), chaque sens possible d'un mot est associé à une fourmilière. Les fourmilières produisent des fourmis. Ces fourmis se déplacent dans le graphe à la recherche d'*énergie* puis la rapportent à leur fourmilière mère qui pourra alors créer de nouvelles fourmis. Pour une fourmi, un nœud peut être : (1) *la fourmilière maison* où elle est née ; (2) *une fourmilière ennemie* qui correspond à un autre sens du même mot ; (3) *une fourmilière potentiellement amie*, toutes celles qui ne sont pas ennemies ; (4) un *nœud qui n'est pas une fourmilière*, les nœuds normaux.

Par exemple, dans la figure 1, pour une fourmi née dans la fourmilière 19, le nœud 18 est un ennemi comme il a

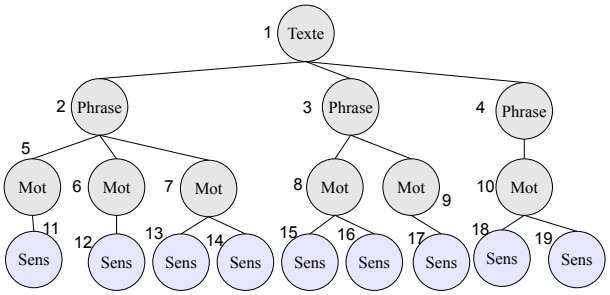

FIGURE 1 – Environnement utilisé dans cette expérience : texte, phrases et mots correspondent aux nœuds dits normaux 1 à 10, un sens de mot correspond à à une fourmilière (nœuds 11 à 19).

le même père (10), les fourmilières potentiellement amies sont les nœuds 11 à 17 et les nœuds normaux sont les nœuds 1 à 10.

Les déplacements des fourmis se déroulent en fonction des scores locaux (cf. section 2.2), de la présence d'énergie, et du passage des autres fourmis (Les fourmis laissent des traces sur les arcs où elles passent sous la forme de *phéromone*). Une fois arrivée sur la fourmilière d'un autre terme, une fourmi peut choisir de revenir directement à sa fourmilière mère. Elle établit alors, entre les deux fourmilières, un pont que les autres fourmis sont, à leur tour, susceptibles d'emprunter et de renforcer grâce à leur phéromone. Ce renforcement a lieu si les informations lexicales conduisent les autres fourmis à emprunter le pont et disparaît dans le cas inverse. Ainsi, les fourmis établissent de nombreux liens entre fourmilières de sens compatibles.

Les ponts correspondent ainsi à des interprétations de la phrase. L'émergence de tels circuits dans le graphe contribue à la monopolisation des ressources de la colonie (fourmis et énergie) et à l'épuisement des ressources associées aux autres fourmilières (ces cas correspondent donc aux sens incompatibles dans le contexte et avec les ressources considérés).

4.2.2 Détails de l'algorithme

Énergie Au début de la simulation, le système possède une certaine énergie qui est répartie équitablement sur chacun des nœuds. Les fourmilières utilisent celle qu'elles possèdent pour fabriquer des fourmis avec une probabilité fonction de cette même énergie et suivant une courbe sigmoïde (cf. fig. 2). On peut remarquer que l'utilisation de cette fonction permet aux fourmilières qui n'ont plus d'énergie de fabriquer quelques fourmis supplémentaires (et ainsi d'avoir une quantité d'énergie négative). L'idée est de leur donner une dernière chance au cas où ces fourmis, trouvant des informations lexicales pertinentes, rapportent de l'énergie et relancent la production de fourmis.

Les fourmis ont une durée de vie (nombre de cycles identique pour toutes et paramétré (cf. tableau 4.2.2)). Lorsqu'une fourmi meurt, l'énergie qu'elle porte ainsi que l'énergie utilisée par la fourmilière pour la produire est déposée sur le nœud où elle se trouve. Il n'y a donc ni perte ni apport d'énergie à aucun moment que ce soit. Si on excepte l'emprunt à la nature que peuvent faire de façon très limitée les fourmilières, le système fonctionne complètement en vase clos. La quantité d'énergie est un élément fondamental de la convergence du système vers une solution. En effet, puisque l'énergie globale est limitée, les fourmilières sont en concurrence les unes avec les

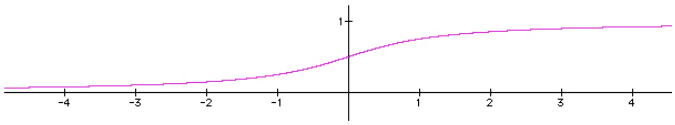

FIGURE 2 – Courbe de la fonction sigmoïde $\frac{arctan(x)}{\pi} + \frac{1}{2}$ qui permet de calculer la probabilité de la naissance d'une fourmi à partir de la quantité d'énergie présente sur le nœud.

autres et seules des alliances peuvent permettre de faire émerger des solutions.

Phéromone de passage Les fourmis ont deux types de comportement. Elles peuvent soit chercher de l'énergie, soit chercher à revenir à leur fourmilière mère. Lorsqu'elles se déplacent dans le graphe, elles laissent des traces sur les arcs où elles passent sous la forme de phéromone. La phéromone influe sur les déplacements des fourmis qui préfèrent l'éviter lorsqu'elles cherchent de l'énergie et préfèrent la suivre lorsqu'elles tentent de revenir déposer cette énergie à leur fourmilière mère.

Lors d'un déplacement, une fourmi laisse une trace en déposant sur l'arc A traversé une quantité de phéromone $\theta \in \mathbb{R}^+$. On a alors $\varphi_{t+1}(A) = \varphi_t(A) + \theta$.

À chaque cycle, il y a une légère évaporation des phéromones. Cette baisse se fait de façon linéaire jusqu'à la disparition totale de la phéromone. Nous avons ainsi, $\varphi_{c+1}(A) = \varphi_c(A) \times (1 - \delta)$ où δ est la proportion de phéromone qui s'évapore à chaque cycle.

Création, suppression et type de ponts Un pont peut être créé lorsqu'une fourmi atteint une fourmilière potentiellement amie, c.-à-d. lorsqu'elle arrive sur un nœud qui correspond à un sens d'un autre mot que celui de la fourmilière mère. Dans ce cas, la fourmi évalue non seulement les nœuds liés à cette fourmilière mais aussi le nœud correspondant à sa fourmilière mère. Si ce dernier est sélectionné, il y a création d'un pont entre les deux fourmilières. Ce pont est ensuite considéré comme un arc standard par les fourmis, c.-à-d. que les nœuds qu'il lie sont considérés comme voisins. Si le pont ne porte plus de phéromone, il disparaît.

Odeur L'odeur d'une fourmilière est la représentation vectorielle que nous avons introduite dans la partie 2.2.2. Elle correspond donc à la définition du sens sous forme de vecteur de nombres entiers. Chaque fourmi née dans cette fourmilière porte la même odeur, le même vecteur. Lors de son déplacement sur les nœuds normaux du graphe, une fourmi propage son vecteur. Le vecteur V(N) porté par un nœud normal N est modifié lors du passage d'une fourmi. La fourmi dépose une partie de son vecteur, un pourcentage des composantes prises au hasard qui remplace la même quantité d'anciennes valeurs elles aussi choisies au hasard.

Cette propagation intervient dans le déplacement des fourmis. Laisser une partie de son vecteur, c'est laisser une trace de passage. Ainsi plus un nœud est proche d'une fourmilière plus il y a de chance que les fourmis de cette fourmilière y soient passées. Ce phénomène permet aux fourmis de revenir à leur fourmilière, ou éventuellement de se tromper et de se diriger vers des fourmilières amies. Cette erreur est ainsi potentiellement bénéfique puisqu'elle

peut permettre de créer un pont entre les deux fourmilières (cf. 4.2.2). En revanche, lorsqu'une fourmi se trouve sur une fourmilière, le vecteur n'est pas modifié. Ces nœuds conservent ainsi un vecteur constant tout au long de la simulation.

La table suivante présente les paramètres, les notations et les valeurs utilisées dans l'algorithme présenté et expérimenté ici. Cet article ne présente pas les expériences réalisées pour trouver ces valeurs.

Notation	Description	Valeurs
F_A	Fourmilière correspondant au sens A	na
$V(X)$	Vecteur odeur associé à X. X est un nœud ou une fourmi	na
f_A	Fourmi née dans la fourmilière F_A	na
E_f	Énergie utilisée par une fourmilière pour produire une fourmi	na
$E(X)$	Énergie possédée par X. X est un nœud ou une fourmi	na
E_{max}	Énergie maximale que peut porter une fourmi	5
$\varphi(A)$	Quantité de phéromone sur l'arc A	na
θ	Phéromone déposée par une fourmi lors de la traversée d'un arc	1
δ	Évaporation de la phéromone entre chaque cycle	20%
$Eval_f(X)$	Évaluation de X selon la fourmi f. X est un arc ou un nœud	na
$Eval_f(N, A)$	Évaluation du nœud N en passant par l'arc A selon la fourmi f	na
	Nombre de cycles de la simulation	100
	Quantité initiale d'énergie sur chaque nœud	20
	Durée de vie d'une fourmi	10
	Énergie prise par une fourmi lorsqu'elle arrive sur un nœud	1
	Longueur du vecteur odeur	50
	Quantité du vecteur odeur modifié par une fourmi lorsqu'elle arrive sur un nœud	10%

4.2.3 Déroulement de l'algorithme

L'algorithme consiste en une itération potentiellement infinie de cycles. À tout moment, la simulation peut être interrompue et l'état courant observé. Durant un cycle, on effectue les tâches suivantes : (1) éliminer les fourmis trop vieilles (la durée de vie est un paramètre) ; (2) pour chaque fourmilière, solliciter la production d'une fourmi (une fourmi peut ou non voir le jour, de façon probabiliste) ; (3) pour chaque arc, diminuer le taux de phéromone (évaporation des traces) ; (4) pour chaque fourmi : déterminer son mode (recherche d'énergie, retour à la fourmilière, le changement est fait de manière probabiliste) et la déplacer. Créer un pont interprétatif le cas échéant ; (5) calculer les conséquences du déplacement des fourmis (sur l'activation des arcs et l'énergie des nœuds).

Les déplacements d'une fourmi sont aléatoires mais influencés par son environnement. Lorsqu'une fourmi est sur un nœud, elle estime tous les nœuds voisins et tous les arcs qui les lient. La probabilité d'emprunter un arc A_j pour aller à un nœud N_i est $P(N_i, A_j) = max(\frac{Eval_f(N_i,A_j)}{\sum_{k=1,l=1}^{k=n,l=m} Eval_f(N_k,A_l)}, \epsilon)$ où $Eval_f(N, A)$ est l'évaluation du nœud N en prenant l'arc A, c.-à-d. la somme de $Eval_f(N)$ et de $Eval_f(A)$. ϵ permet à certaines fourmis de choisir des destinations évaluées comme improbables mais qui permettraient d'atteindre des informations lexicales et des ressources qui s'avèreraient intéressantes ensuite.

Une fourmi qui vient de naître (c.-à-d. être produite par sa fourmilière) part à la recherche d'énergie. Elle est attirée par les nœuds qui portent beaucoup d'énergie ($Eval_f(N) = \frac{E(N)}{\sum_0^m E(N_i)}$) et évite les arcs qui portent beaucoup de phéromone ($Eval_f(A) = 1 - \varphi(A)$) afin de permettre l'exploration de plus de solutions. Elle continue à collecter de l'énergie jusqu'au cycle où un tirage aléatoire avec la probabilité $P(retour) = \frac{E(f)}{E_{\max}}$ la fera passer en mode retour. Dans ce mode, elle va (statistiquement) suivre les arcs avec beaucoup de phéromone ($Eval_f(A) = \varphi(A)$) et vers les nœuds dont l'odeur est proche de la leur ($Eval_f(N) = \frac{Lesk(V(N),V(f_A))}{\sum_{i=1}^{i=k} Lesk(V(N_i),V(f_A))}$).

5 Évaluation

Nous avons testé notre méthode sur le corpus de la tâche *gros grain* de la campagne d'évaluation *Semeval 2007* (Navigli *et al.*, 2007) dans laquelle les organisateurs fournissent un inventaire de sens plus grossiers que ceux de WordNet. Pour chaque terme, les sens considérés comme proches (par exemple, "*neige/précipitation*" et "*neige/couverture*" ou "*porc/animal*" et "*porc/viande*") sont groupés. Le corpus est composé de 5 textes de genres divers (journalisme, critique littéraire, voyage, informatique, biographies) dont il faut annoter les 2269 mots. Le nombre moyen de sens par mot est de 6,19 ; ramené à 3,1 pour l'inventaire de sens grossiers. Les compétiteurs étaient libres de se servir de cet inventaire (sens grossiers connus *a priori*) ou non (sens grossiers connus *a posteriori*). Dans le premier cas, le nombre de choix à faire pour chaque mot est réduit et la tâche moins compliquée. Dans le second cas, les sens annotés sont jugés corrects s'ils sont dans le bon groupement, une sorte d'erreur acceptable. Notre objectif est de tester un système en vue d'une utilisation dans un cadre applicatif réel or l'inventaire de sens grossiers n'est disponible que pour les 2269 mots utilisés dans le corpus d'évaluation, nous ne l'utilisons donc pas. Dans les expériences présentées ici, nous nous situons ainsi dans un cas de sens connus *a posteriori*. Les résultats sont analysés par les formules classiques :

$$\text{Précision } P = \frac{sens\ correctement\ annotés}{sens\ annotés} \qquad \text{Rappel } R = \frac{sens\ correctement\ annotés}{sens\ à\ annoter} \qquad \text{F-mesure } F = \frac{2 \times P \times R}{P+R}$$

Dans le corpus, les mots sont annotés avec leur partie du discours (verbe, nom, adverbe, adjectif). À partir de ces informations, nous construisons l'environnement des fourmis : un nœud au niveau du texte, un nœud pour chaque phrase, un nœud pour chaque mot et une fourmilière pour chaque sens (voir fig. 1). À la fin d'un cycle, le sens sélectionné pour chaque mot correspond à la fourmilière qui a la plus grande quantité d'énergie.

5.1 Exécution de l'algorithme

L'algorithme à colonies de fourmis garantit la réalisation d'un choix entre les différentes possibilités pour chaque terme. Ainsi, 100% du corpus est annoté et P=R=F puisque les sens annotés sont égaux aux sens à annoter (P=R) et dans ce cas $F = \frac{2 \times P \times P}{P+P} = \frac{2 \times P^2}{2P} = P$. De plus, un algorithme à colonies de fourmis est un algorithme stochastique, il ne sélectionne donc pas exactement les mêmes sens à chaque exécution ni même à chaque cycle. Nous avons exécuté cet algorithme des centaines de fois et avons noté qu'après 70-80 cycles, les résultats restaient globalement constants comme l'illustre la figure suivante.

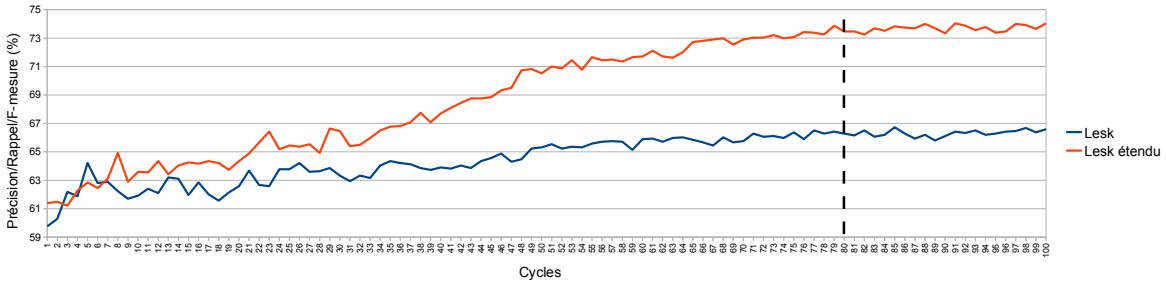

FIGURE 3 – Évolution de la précision/rappel/F-mesure dans les 100 cycles d'une exécution de l'algorithme à colonies de fourmis utilisé avec l'algorithme local Lesk et avec l'algorithme local Lesk étendu

5.2 Comparaison d'exécutions

De la même manière que les résultats évoluent entre deux cycles, les résultats peuvent être différents entre deux exécutions. Pour donner une idée de cette différence, nous avons répété notre expérience, arrêtée au bout de 100 cycles, sur chacun des algorithmes locaux, 100 fois. La table suivante en présente les résultats. Nous obtenons seulement $2,95\%$ d'écart entre le meilleur et le moins bon résultat (soit 67 termes mal annotés sur 2269) pour *Lesk étendu* et 3,39% (soit 77 termes mal annotés sur 2269) pour *Lesk*.

Algorithme local	Minimum	Maximum	Moyenne	Médiane	Étendue	Écart-type
Lesk	64,43	67,83	66,34	66,35	3,39	0,66
Lesk étendu	72,54	75,5	74,01	74,04	2,95	0,58

5.3 Comparaisons avec l'algorithme exhaustif

À titre de comparaison avec notre approche, nous présentons les résultats obtenus par l'algorithme global exhaustif (Banerjee & Pedersen, 2002). Nous avons choisi comme contexte la phrase, excluant *de facto* les phrases d'un mot (au nombre de quatre, soit moins de $0,002\%$ du corpus). Pour des raisons calculatoires, nous avons également exclu les phrases de plus de 10 milliards de combinaisons. Nous pouvons voir que seulement $77,3\%$ du corpus a

Algorithme global	Algorithme local	Étiquetés	Précision	Rappel	F-mesure	Temps
Calcul exhaustif	Lesk	77,30	69,21	53,50	60,35	$\approx$ 40h
	Lesk étendu	77,30	77,82	60,16	67,86	$\approx$ 300h
Fourmis	Lesk	100,0	64,43 - 67,83	64,43 - 67,83	64,43 - 67,83	$\approx$ 3m
	Lesk étendu	100,0	72,54 - 75,5	72,54 - 75,5	72,54 - 75,5	$\approx$ 8m

été annoté au prix d'une durée de plusieurs heures incompatible avec des applications en temps réel [4].

Pour les deux algorithmes locaux, la F-mesure est clairement supérieure à celle du calcul brut pour un temps nettement moins long (800 fois plus court pour Lesk et 2250 fois pour Lesk étendu). Le tableau suivant présente pour les mêmes exécutions les résultats sur les différentes sous-parties du corpus : A, la partie annoté par les 2 algorithmes globaux et B celle qui n'est annotée que par l'algorithme fourmis. Sur la partie A, les fourmis sont, comme on pouvait s'en douter, légèrement en dessous de l'algorithme exhaustif et leur meilleur résultat s'explique par la possibilité d'annoter la sous-partie B.

Pour conclure cette évaluation, nous avons comparé nos résultats avec les résultats obtenus par les différents systèmes qui participaient à la campagne Semeval 2007. Avec Lesk étendu, nous serions arrivés $8^{ème}$/15 en tenant compte de tous les participants, $5^{ème}$/8 sur ceux qui ne connaissent pas *a priori* les sens grossiers, 1^{er}/7 sur les approches non supervisées. Ces résultats sont très encourageants vu les temps de calcul (aucun article des participants n'aborde ce point), les possibilités d'extension qu'offrent les algorithmes à fourmis et la simplicité des algorithmes locaux envisagés ici.

4. Expériences réalisées sur des processeurs Intel Xeon X5550, 4 cœurs à 2.66Ghz (durées converties en temps monoprocesseurs).

Algorithme local	Sous-corpus	Algorithme global	Étiquetés	Rappel	Différentiel
Lesk	A + B	Exhaustif	77,30	53,50	
		Fourmis	100,0	64,43 - 67,83	+ 10,93 à + 14,33
	A	Exhaustif	100,0	69,21	
		Fourmis	100,0	65,45 - 68,99	- 3,76 à - 0,22
	B	Exhaustif	00,00	00,00	
		Fourmis	100,0	60,97 - 63,88	+ 60,97 à + 63,88
Lesk étendu	A + B	Exhaustif	77,30	60,16	
		Fourmis	100,0	72,54 - 75,5	+ 12,38 à + 15,34
	A	Exhaustif	100,0	77,82	
		Fourmis	100,0	74,69 - 77,25	- 3,13 à - 0,57
	B	Exhaustif	00,00	00,00	
		Fourmis	100,0	65,24 - 69,52	+ 65,24 à + 69,52

6 Conclusions et Perspectives

Dans cet article, nous avons présenté un algorithme à colonies de fourmis destiné à la désambiguïsation lexicale et basé sur des mesures de proximité sémantique. Cet algorithme, non supervisé, est volontairement simple puisqu'il n'utilise qu'une seule ressource lexicale (WordNet) et aucune analyse morphologique ou morpho-syntaxique. Il permet pourtant de choisir un sens, pour chaque mot d'un texte, d'une manière plus rapide que l'algorithme exhaustif et en atteignant une bonne F-mesure pour un système non supervisé. Nous considérons ces résultats comme une ligne de base (baseline) à partir de laquelle nous allons poursuivre nos recherches. Outre l'ajout d'informations morphologiques et/ou syntaxiques, nous travaillons actuellement sur la combinaison de mesures locales et l'utilisation de WordNet dans l'environnement des fourmis. Nos travaux portent également sur d'autres algorithmes locaux et leur impact sur l'utilisation dans d'autres langues notamment flexionnelles. Enfin, nous travaillons à la comparaison des algorithmes à colonies de fourmis avec d'autres algorithmes globaux comme les algorithmes génétiques ou le recuit simulé.

Références

AGIRRE E. & EDMONDS P. (2006). *Word Sense Disambiguation : Algorithms and Applications (Text, Speech and Language Technology)*. Secaucus, NJ, USA : Springer-Verlag New York, Inc.

BANERJEE S. & PEDERSEN T. (2002). An adapted lesk algorithm for word sense disambiguation using wordnet. In *the Third International Conference on Intelligent Text Processing and Computational Linguistics, CICLing 2002*, Mexico City.

BONABEAU É. & THÉRAULAZ G. (2000). L'intelligence en essaim. *Pour la science*, (271), 66–73.

COWIE J., GUTHRIE J. & GUTHRIE L. (1992). Lexical disambiguation using simulated annealing. In *COLING 1992, International Conference on Computational Linguistics*, volume 1, p. 359–365, Nantes, France.

CRAMER I., WANDMACHER T. & WALTINGER U. (2010). *WordNet : An electronic lexical database*, chapter Modeling, Learning and Processing of Text Technological Data Structures. Springer.

DEERWESTER S. C., DUMAIS S. T., LANDAUER T. K., FURNAS G. W. & HARSHMAN R. A. (1990). Indexing by latent semantic analysis. *Journal of the American Society of Information Science*, **41**(6).

DENEUBOURG J.-L., GROSS S., FRANKS N. & PASTEELS J.-M. (1989). The blind leading the blind : Modeling chemically mediated army ant raid patterns. *Journal of Insect Behavior*, **2**, 719–725.

DORIGO & STÜTZLE (2004). *Ant Colony Optimization*. MIT-Press.

DORIGO M. & GAMBARDELLA L. (1997). Ant colony system : A cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, **1**, 53–66.

DROGOUL A. (1993). When ants play chess (or can strategies emerge from tactical behaviors). In *Maamaw'1993*.

FELLBAUM C. (1998). *WordNet : An Electronic Lexical Database (Language, Speech, and Communication)*. The MIT Press.

GALE W., CHURCH K. & YAROWSKY D. (1992). One sense per discourse. In *Fifth DARPA Speech and Natural Language Workshop*, p. 233–237, Harriman, New-York, États-Unis.

GELBUKH A., SIDOROV G. & HAN S. Y. (2003). Evolutionary approach to natural language word sense disambiguation through global coherence optimization. *WSEAS Transactions on Communications*, **2**(1), 11–19.

GUINAND F. & LAFOURCADE M. (2009). *Fourmis Artificielles 2. Nouvelles Directions pour une Intelligence Collective*, chapter Fourmis Artificielles et Traitement de la Langue Naturelle, p. 225–267. Lavoisier.

IDE N. & VÉRONIS J. (1998). Word sense disambiguation : the state of the art. *Computational Linguistics*, **28**(1), 1–41.

LESK M. (1986). Automatic sense disambiguation using machine readable dictionaries : how to tell a pine cone from an ice cream cone. In *Proceedings of the 5th annual international conference on Systems documentation*, SIGDOC '86, p. 24–26, New York, NY, USA : ACM.

N. MONMARCHE, F. GUINAND & P. SIARRY, Eds. (2009). *Fourmis Artificielles et Traitement de la Langue Naturelle*. Prague, Czech Republic : Lavoisier.

NAVIGLI R. (2009). Word sense disambiguation : a survey. *ACM Computing Surveys*, **41**(2), 1–69.

NAVIGLI R. & LAPATA M. (2010). An experimental study of graph connectivity for unsupervised word sense disambiguation. *IEEE Trans. Pattern Anal. Mach. Intell.*, p. 678–692.

NAVIGLI R., LITKOWSKI K. C. & HARGRAVES O. (2007). Semeval-2007 task 07 : Coarse-grained english all-words task. In *Proceedings of the Fourth International Workshop on Semantic Evaluations (SemEval-2007)*, p. 30–35, Prague, Czech Republic : Association for Computational Linguistics.

PEDERSEN T., BANERJEE S. & PATWARDHAN S. (2005). *Maximizing Semantic Relatedness to Perform Word Sense Disambiguation*. Research Report UMSI 2005/25, University of Minnesota Supercomputing Institute.

PONZETTO S. P. & NAVIGLI R. (2010). Knowledge-rich word sense disambiguation rivaling supervised systems. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, ACL '10, p. 1522–1531, Stroudsburg, PA, USA : Association for Computational Linguistics.

ROUQUET D., FALAISE A., SCHWAB D., BOITET C., BELLYNCK V., NGUYEN H.-T., MANGEOT M. & GUILBAUD J.-P. (2010). *Rapport final de synthèse, passage à l'échelle et implémentation : Extraction de contenu sémantique dans des masses de données textuelles multilingues*. Rapport interne, Agence Nationale de la Recherche.

SCHWAB D. (2005). *Approche hybride - lexicale et thématique - pour la modélisation, la détection et l'exploitation des fonctions lexicales en vue de l'analyse sémantique de texte.* PhD thesis, Université Montpellier 2.

SCHWAB D. & LAFOURCADE M. (2007). Lexical functions for ants based semantic analysis. In *ICAI'07- The 2007 International Conference on Artificial Intelligence*, Las Vegas, Nevada, USA.

VASILESCU F., LANGLAIS P. & LAPALME G. (2004). Evaluating variants of the lesk approach for disambiguating words. In *Proceedings of LREC 2004, the 4th International Conference On Language Resources And Evaluation*, p. 633–636, Lisbon, Portugal.