Désambiguïsation lexicale de textes : efficacité qualitative et temporelle d'un algorithme à colonies de fourmis

Didier Schwab — Jérôme Goulian — Andon Tchechmedjiev

Univ. Grenoble Alpes, Laboratoire d'Informatique de Grenoble équipe GETALP
41 rue des mathématiques, BP 53 38041 Grenoble Cedex 9

{Didier.Schwab,Jerome.Goulian,Andon.Tchechmedjiev}@imag.fr, http://getalp.imag.fr/WSD/

RÉSUMÉ. *Dans cet article, nous présentons la notion d'algorithme local et d'algorithme global pour la désambiguïsation lexicale de textes. Un algorithme local permet de calculer la proximité sémantique entre deux objets lexicaux. L'algorithme global permet de propager ces mesures locales à un niveau supérieur. Nous nous servons de cette notion pour confronter un algorithme à colonies de fourmis à d'autres méthodes issues de l'état de l'art, un algorithme génétique et un recuit simulé. En les évaluant sur un corpus de référence, nous montrons que l'efficacité temporelle des algorithmes à colonies de fourmis rend possible l'amélioration automatique du paramétrage et, en retour, leur amélioration qualitative. Enfin, nous étudions plusieurs stratégies de fusion tardive des résultats de nos algorithmes pour améliorer leurs performances.*

ABSTRACT. *In this article, we present the notions of local and global algorithms, for the word sense disambiguation of texts. A local algorithm allows to calculate the semantic similarity between two lexical objects. Global algorithms propagate local measures at the upper level. We use this notion to compare an ant colony algorithm to other methods from the state of the art: a genetic algorithm and simulated annealing. Through their evaluation on a reference corpus, we show that the run-time efficiency of the ant colony algorithm makes the automated estimation of parameters possible and in turn the improvement of the quality results as well. Last, we study several late classifier fusion strategies over the results to improve the performance.*

MOTS-CLÉS : *désambiguïsation lexicale fondée sur des similarités, algorithmes locaux/globaux, algorithmes à colonies de fourmis, algorithmes stochastiques d'optimisation.*

KEYWORDS: *Word Sense Disambiguation based on similarity measures, local/global algorithms, Ant Colony Algorithms, Stochastic optimization algorithms.*

1. Introduction

La désambiguïsation lexicale est une tâche centrale pour le traitement automatique des langues. Elle peut en effet permettre d'améliorer de nombreuses applications 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 prédéfini. 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, contrairement aux secondes. Nous pensons que cette typologie devrait être raffinée. En effet, de nombreuses approches utilisent, généralement sans le dire, un algorithme local et un algorithme global. L'algorithme local permet de donner une mesure de la proximité sémantique entre deux objets lexicaux (sens, mots, constituants, etc.) tandis que l'algorithme global permet de propager les mesures locales à un niveau supérieur. Cette double typologie nous paraît pourtant centrale et permet de mieux caractériser les propriétés des systèmes de désambiguïsation lexicale. Ainsi un système peut être constitué d'un algorithme local plus ou moins supervisé et d'un algorithme global lui aussi plus ou moins supervisé.

Dans cet article, nous présentons cette notion et montrons qu'elle permet de mieux comprendre certaines propriétés et performances des systèmes mais aussi de mieux évaluer une certaine limite de la non supervision complète. Nous nous servons ensuite de cette notion pour confronter un algorithme à colonies de fourmis (ACA) à d'autres méthodes issues de l'état de l'art, un algorithme génétique (GA) et un recuit simulé (SA). En les évaluant sur un corpus de référence, nous montrons que l'efficacité temporelle des algorithmes à colonies de fourmis rend possible l'amélioration automatique du paramétrage et, en retour, l'amélioration qualitative des résultats. Enfin, nous étudions plusieurs stratégies de fusion tardive des résultats de nos algorithmes et montrons que l'une d'elle atteint le niveau de la référence premier sens et rivalise avec les moins bons algorithmes supervisés.

2. Désambiguïsation lexicale : bref état de l'art

Schématiquement, une tâche de désambiguïsation lexicale consiste à choisir quel est le sens le plus approprié pour chaque mot d'un texte. Comme rappelé en introduction, de nombreux travaux existent sur le sujet. Le lecteur pourra consulter (Ide et Véronis, 1998) pour les travaux antérieurs à 1998 et (Agirre et Edmonds, 2006) ou (Navigli, 2009) pour un état de l'art complet.

2.1. *Approches supervisées*

Les approches supervisées (ou d'apprentissage automatique) reposent sur l'utilisation d'un nombre important de ressources annotées. Les données d'apprentissage

permettent alors de créer un classifieur qui peut déterminer le sens le plus adapté d'un mot dans un contexte donné. De nombreuses méthodes supervisées comme les listes et arbres de décision, les classifications naïves bayésiennes, les réseaux de neurones, les approches de type plus proches voisins, les machines à vecteurs de support, etc. ont été appliquées au problème de la désambiguïsation lexicale. Même si les méthodes supervisées tendent à donner de meilleurs résultats (sur l'anglais) que les méthodes non supervisées, à la fois en termes de vitesse et de qualité, leur principal désavantage est qu'elles nécessitent une grande quantité de données annotées manuellement pour un inventaire de sens donné, une langue donnée ou pour un domaine spécifique donné (sport, finance...). Cette constatation, que nous partageons avec Navigli et Lapata (2010), nous a conduits à nous intéresser plus particulièrement à des approches non supervisées.

2.2. *Approches non supervisées*

Les approches non supervisées n'utilisent pas de corpus annotés. Certaines de ces méthodes utilisent des corpus non annotés pour construire des vecteurs de mots ou des graphes de cooccurrences tandis que d'autres utilisent des sources de connaissance externes (dictionnaires, thésaurus, bases lexicales...). 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 (au sens mathématique du terme) et donc avoir une valeur entre 0 et 1, des distances, et donc respecter leurs trois propriétés (séparation, symétrie et inégalité triangulaire) ou plus généralement une valeur positive non bornée. Parmi ces mesures, on peut citer les mesures ou distances entre vecteurs (vecteurs conceptuels (Schwab, 2005), LSA (Deerwester *et al.*, 1990)), les mesures fondées sur une distance taxonomique dans un réseau lexical (nombre d'arcs dans le graphe entre deux sens), qu'elles utilisent toutes (Hirst et St-Onge, 1998) ou une partie des relations du réseau (Rada *et al.*, 1989 ; Leacock et Chodorow, 1998 ; Wu et Palmer, 1994), les mesures basées sur le contenu d'information (Resnik, 1995 ; Lin, 1998 ; Seco *et al.*, 2004) ou encore des mesures hybrides combinant plusieurs de ces approches (Li *et al.*, 2003 ; Pirró et Euzenat, 2010). Le lecteur pourra consulter (Pedersen *et al.*, 2005 ; Cramer *et al.*, 2010 ; Navigli, 2009) pour un panorama plus complet. En désambiguïsation lexicale, ces mesures sont utilisées de façon locale entre deux sens de mots, et sont ensuite appliquées à un niveau global. Dans cet article, nous utilisons une telle mesure comme score local (voir section 3.1.2).

2.3. *Approches semi-supervisées*

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, 2009).

3. Algorithmes locaux et globaux pour la désambiguïsation lexicale fondée sur des similarités

De nombreux systèmes de désambiguïsation lexicale reposent sur la notion d'algorithme local et d'algorithme global. L'algorithme local permet de donner une mesure de la proximité sémantique entre deux objets lexicaux (sens, mots, constituants, etc.) tandis que l'algorithme global permet de propager les mesures locales à un niveau supérieur.

3.1. *Algorithmes locaux*

Les algorithmes locaux sont les méthodes présentées dans la section 2.2. En anglais, langue de notre corpus d'évaluation, elles sont souvent utilisées sur WordNet.

3.1.1. *Bases lexicales utilisées en désambiguïsation lexicale : WordNet et BabelNet*

WordNet (Fellbaum, 1998) est une base lexicale pour l'anglais très largement utilisée dans le cadre de la désambiguïsation lexicale. Elle est organisée en ensembles de synonymes (*synsets*) auxquels sont associées leurs parties du discours et les relations sémantiques qu'ils entretiennent avec d'autres *synsets* (antonymes, hyponymes, méronymes...) ainsi qu'une définition. La version actuelle de WordNet, la 3.0, contient plus de 155 000 mots pour 117 000 synsets.

Il existe depuis peu BabelNet (Navigli et Ponzetto, 2012), une base alignant les sens de WordNet à ceux de Wikipedia ce qui permet de définir les termes avec des définitions dans plusieurs langues. Nous commençons à l'exploiter par ailleurs mais l'algorithme local de cet article se base seulement sur la partie WordNet.

3.1.2. *Notre algorithme local : inspiré par Lesk*

L'algorithme local utilisé dans cet article est une variante de l'algorithme de Lesk (Lesk, 1986). Proposé il y a plus de vingt-cinq ans, cet algorithme 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 et Navigli, 2010).

Notre algorithme local exploite les liens présents dans WordNet. Au lieu d'utiliser uniquement la définition d'un sens, elle utilise également la définition des différents

sens qui lui sont liés[1]. Cette idée est similaire à celle de Banerjee et Pedersen (2002)[2]. Cet algorithme local est nommé dans la suite $Lesk_{ext}$.

Pour des raisons d'efficacité (Schwab *et al.*, 2011 ; Schwab *et al.*, 2012), les définitions sont transformées en vecteurs d'entiers qui sont triés, chaque entier correspondant à un mot. Notre algorithme de propagation à colonies de fourmis utilise également ces pseudodéfinitions créées à la volée (voir partie 5.2.2.3). L'implantation et le dictionnaire qui permettent d'utiliser $Lesk_{ext}$ peuvent être trouvés sur le site du sous-groupe WSD du GETALP[3] et en particulier sur la page dédiée à cet article[4].

Récemment, Miller *et al.* (2012) ont présenté un algorithme local qui obtient de meilleurs résultats que notre $Lesk_{ext}$ (voir section 6.6). Il s'agit d'une nouvelle extension des définitions fondée sur leur voisinage dans un corpus. Cette solution semble très simple à mettre en œuvre et nous étudions actuellement cette possibilité. Nous rappelons au lecteur que cet article concerne les algorithmes globaux et que nous aurions pu choisir n'importe quel algorithme local pour l'illustrer.

3.2. *Algorithmes globaux stochastiques pour la désambiguïsation lexicale*

Un algorithme global propage à un niveau supérieur les mesures locales. Nous présentons dans les section suivantes chacun des algorithmes étudiés. Le lecteur trouvera en annexes leur description complète en langage algorithmique.

3.2.1. *Approche exhaustive*

La méthode de propagation la plus directe est la recherche exhaustive utilisée par exemple dans (Banerjee et 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 (dans laquelle seize mots ont une entrée dans notre dictionnaire) tirée du corpus d'évaluation que nous utilisons dans la partie 6.1, « *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*› et ‹*air*› ont neuf sens, ‹*paint*› quatre, ‹*be*›, ‹*point*› et ‹*figure*› treize, ‹*flat*› dix-sept, ‹*very*› et ‹*often*› deux, ‹*foot*› et ‹*ground*› onze, ‹*look*› dix, ‹*stand*› douze, ‹*at all*› et ‹*downwards*› un et ‹*hang*› quinze. 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

1. L'ensemble des relations sémantiques présentes dans WordNet est utilisé.
2. Banerjee et Pedersen (2002) introduisent 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.
3. `http://getalp.imag.fr/WSD/`
4. `http://getalp.imag.fr/static/wsd/GETALP-WSD-ACA/TAL-54-1/index.xhtml`

nécessite des dizaines voire des centaines d'opérations) que peuvent théoriquement effectuer 542 processeurs Intel Xeon Phi Coprocessor 5110P (1,053 GHz, 60 cœurs, 240 fils d'exécutions simultanées, 2 700 dollars l'unité), les plus rapides existant à la mi-2013 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 approches sont possibles. Les premières, dites approches complètes, tentent de réduire la combinatoire en utilisant des techniques d'élagage et des heuristiques de choix. Dans le cadre de la désambiguïsation lexicale, c'est le cas par exemple de l'approche proposée par Hirst et St-Onge (1998), fondée sur les chaînes lexicales (une mesure de similarité sémantique à base de distance taxonomique exploitant l'ensemble des relations de WordNet), combinant des restrictions[5] lors de la construction de la chaîne lexicale globale avec une heuristique de choix gloutonne. Selon Navigli (2009) le problème majeur d'une telle approche est son manque de précision du fait de la stratégie gloutonne employée. Différentes améliorations ont toutefois été proposées parmi lesquelles on peut notamment citer Silber et McCoy (2000). D'autres approches complètes intéressantes ont été menées dans le cadre de la désambiguïsation lexicale non supervisée ; citons en particulier Brody et Lapata (2008).

Les secondes méthodes, dites approches incomplètes parce qu'elles n'explorent qu'une partie de l'espace de recherche, utilisent des heuristiques permettant de se guider vers des zones de l'espace de recherche semblant plus prometteuses. Ces heuristiques s'appuient en général sur des probabilités et les choix sont réalisés de façon stochastique, c'est ce qui nous intéresse ici.

On peut alors distinguer deux grandes familles de méthodes :

– les approches par voisinage (de nouvelles configurations sont créées à partir de configurations existantes) parmi lesquelles on trouve des approches issues de l'intelligence artificielle comme les algorithmes génétiques ou des méthodes d'optimisation comme le recuit simulé ;

– les approches constructives (de nouvelles configurations sont générées par ajout itératif d'éléments de solution aux configurations en cours de construction) parmi lesquelles on trouve par exemple les algorithmes à base de fourmis

3.2.2. *Cadre de notre étude, hypothèse de travail*

L'objectif de cet article est de comparer notre algorithme à colonies de fourmis (approche incomplète constructive) à d'autres approches incomplètes. Nous avons choisi de confronter dans un premier temps notre approche à deux approches par voisinage

5. De telles restrictions, pour diminuer le nombre de combinaisons à examiner, peuvent également être fondées sur l'utilisation de corpus comme par exemple la recherche des chaînes lexicales compatibles (Gale *et al.*, 1992 ; Vasilescu *et al.*, 2004). Ces approches rentrent alors, de ce fait, dans le cadre de la désambiguïsation lexicale supervisée.

ayant été utilisées dans le cadre de la désambiguïsation lexicale non supervisée : les algorithmes génétiques (Gelbukh *et al.*, 2003) et le recuit simulé (Cowie *et al.*, 1992).

Notre hypothèse de travail, dans cet article, est en effet de considérer comme fenêtre d'analyse, le texte. Ce choix est également effectué par Cowie *et al.* (1992) pour leurs algorithmes de recuit simulé et par Gelbukh *et al.* (2003) pour leur algorithme génétique ; cette idée étant reprise récemment par Navigli et Lapata (2010). De nombreuses approches, en revanche, toujours pour des raisons calculatoires et parfois sans le dire explicitement, utilisent un contexte plus réduit. Nous y voyons deux problèmes. Le premier est que nous n'avons aucun moyen d'assurer la cohérence entre les sens choisis. Deux sens généralement incompatibles entre eux pourront être choisis car le contexte ne comprend pas le deuxième mot. Par exemple, même avec une fenêtre de six mots avant et six mots après, la phrase « *L'homme est rentré chez lui vers 19 h 30 et a garé sa voiture sur le parking. Le chien était bien en place lorsque l'homme rangea son fusil.* », « *fusil* » n'entre pas en compte pour désambiguïser le terme « *chien* ». Le second problème est qu'un texte conserve généralement une certaine unité sémantique. Par exemple, comme le constatent Gale *et al.* (1992) ou Hirst et St-Onge (1998), un mot utilisé plusieurs fois dans un texte aura généralement le même sens ; cette information ne pouvant pas être exploitée avec un fenêtrage.

C'est en raison de cette présence d'un fenêtrage que nous avons écarté de notre étude des approches telles que *Lesk étendu simplifié*[6] de Miller *et al.* (2012) ou l'application à la désambiguïsation lexicale faite par Mihalcea *et al.* (2004) de l'algorithme de *PageRank* (Brin et Page, 1998)[7], bien que travaillant sur un espace de recherche sous forme de graphe comme les algorithmes à colonies de fourmis.

Par ailleurs, l'objectif de cet article n'est pas ici de comparer entre elles les qualités de ces trois approches incomplètes par rapport à l'optimum. En effet, nous souhaitons à terme nous placer dans le cadre d'applications nécessitant une désambiguïsation en temps réel. Or, nos premières expériences (Schwab *et al.*, 2011) nous ont permis de constater que l'algorithme exhaustif utilisant notre mesure sémantique locale, évalué, faute de mieux étant donnée la combinatoire, sur un contexte réduit (la phrase) n'était en mesure de ne fournir un optimum que dans environ 77 % des cas.

6. Le choix des termes *Lesk simplifié*, *Lesk étendu simplifié* est d'ailleurs certainement peu pertinent. Dans la perspective *algorithme local/global*, ces noms font un mélange peu propice à la compréhension. L'algorithme global compare simplement un sens de mot aux mots dans le contexte du mot à désambiguïser. L'algorithme local peut reposer sur plusieurs matériaux : un dictionnaire de définitions simples, un dictionnaire de définitions étendues grâce à un réseau sémantique, un dictionnaire de définitions étendues grâce à un corpus, etc. Nous préférons le terme de *Lesk-contexte*. Sur la question des performances, *Lesk-contexte* est un algorithme très efficace et rapide mais qui ne nous semble pas offrir de réelles perspectives d'améliorations, en particulier vers la désambiguïsation utilisant des informations de nature multilingue (désambiguïsation d'une langue peu dotée par un inventaire de sens décrit dans d'autres langues à la manière de BabelNet par exemple).

7. Cet algorithme, par ailleurs déterministe, simule une marche aléatoire.

3.2.3. *Présentation générale*

L'objectif des algorithmes étudiés ici est d'assigner à chaque mot w_i d'un texte de *m* mots, l'un de ses sens $w_{i,j}$. La définition du sens j du mot i est notée $d(w_{i,j})$. L'espace de recherche correspond à toutes les combinaisons de sens possibles pour le texte considéré. Ainsi, une configuration C du problème est représentée par un vecteur d'entiers tel que $j = C[i]$ est le sens $w_{i,j}$.

3.3. *Configuration et score global (fonction de coût)*

Ces algorithmes ont besoin d'une mesure pour permettre une évaluation pertinente d'une configuration donnée. Le score du sens sélectionné pour un mot donné peut être exprimé par la somme des scores locaux entre ce sens et les sens sélectionnés de tous les autres mots du contexte considéré. Ainsi, pour évaluer une configuration du problème donné, on peut considérer comme *mesure globale*, comme fonction de coût, la somme des scores de tous les sens sélectionnés des mots du texte :

$$Score(C) = \sum_{i=1}^{m} \sum_{j=i}^{m} Lesk_{ext}(w_{i,C[i]}, w_{j,C[j]}).$$

4. Deux algorithmes stochastiques de l'état de l'art

4.1. *Algorithme génétique pour la désambiguïsation lexicale*

L'algorithme génétique (GA pour *genetic algorithme*), inspiré de Gelbukh *et al.* (2003), peut être découpé en cinq étapes distinctes : initialisation, sélection, croisement, mutation et évaluation. Mis à part l'initialisation, les autres étapes sont exécutées dans cet ordre pour chaque génération de la population.

L'étape d'initialisation consiste en la génération aléatoire d'une population de λ individus (λ configurations du problème). La taille de la population reste inchangée pendant toute l'exécution de l'algorithme.

Pendant l'étape de sélection, le score de chaque individu de la population est calculé. Un taux de croisement (*CR* pour *Crossover Ratio*) est utilisé pour déterminer quels individus de la population seront choisis pour un croisement. La probabilité qu'un individu soit choisi est pondérée en fonction de son score et de celui du meilleur individu de la population. Les individus qui ne sont pas sélectionnés pour un croisement sont clonés dans la nouvelle population. Par ailleurs, le meilleur individu est systématiquement ajouté à la nouvelle population.

L'étape de croisement consiste à trier les individus en fonction de leur score global puis de les croiser. Si le nombre d'individus est impair, l'individu au score le plus bas est ajouté directement dans la nouvelle population. L'opération de croisement est alors appliquée pour chacune des paires d'individus. Les paires sont formées selon l'ordre décroissant des scores globaux de chacun des individus. Les gènes des deux

individus (les sens sélectionnés dans les configurations) sont alors échangés autour de deux pivots choisis aléatoirement (tout ce qui est entre les pivots est échangé).

À l'étape de mutation, chaque individu a ensuite une probabilité de muter (paramètre *MR* (pour *Mutation Rate*)). L'étape de mutation consiste à réaliser *MN* changements aléatoires sur les individus (configurations) concernés par la mutation.

L'étape d'évaluation correspond à l'évaluation du critère de terminaison de l'algorithme, à savoir ici une convergence du score du meilleur individu. Pour vérifier la convergence, un seuil *STH* est utilisé. En d'autres termes, si le score du meilleur individu reste le même pendant *STH* générations, l'algorithme se termine.

4.2. *Recuit simulé pour la désambiguïsation lexicale*

La méthode du recuit simulé (SA pour *simulated annealing*) telle que décrite dans (Cowie *et al.*, 1992) est fondée sur les principes physiques du refroidissement des métaux.

La méthode du recuit simulé travaille sur la même représentation du problème que l'algorithme génétique présenté dans la section précédente. Cependant il ne travaille que sur une unique configuration du problème, choisie aléatoirement au départ. L'algorithme s'exécute en cycles. Chaque cycle est composé de *IN* itérations. Les autres paramètres sont la température initiale T_0[8] et le taux de refroidissement (*ClR* pour *Cooling Rate*, $\in [0;1]$). À chaque itération, une modification aléatoire est réalisée sur la configuration courante C_c pour produire une nouvelle configuration $C_c^{'}$. Étant donné $\Delta E = Score(C_c) - Score(C_c^{'})$, la probabilité $P(A)$ d'acceptation[9] de la configuration $C_c^{'}$ à la place de C_c est :

$$P(A) = \begin{cases} 1 & \text{if } \Delta E < 0 \\ e^{\frac{-\Delta E}{T}} & \text{sinon} \end{cases}$$

La configuration courante peut évoluer vers une configuration ayant un score inférieur afin d'empêcher l'algorithme de converger vers un maximum local et lui permet ainsi d'explorer d'autres parties de l'espace de recherche qui contiendront le maximum global.

Si, à la fin d'un cycle, la configuration courante est restée inchangée par rapport à celle du cycle précédent, l'algorithme se termine. Dans le cas contraire, la température est abaissée de $T \cdot ClR$. En d'autres termes, plus il faut de cycles pour que l'algorithme

8. Le choix de la température initiale passe par le choix de la probabilité d'acceptation initiale. Dans les expériences décrites section 6.3, nous nous plaçons exactement dans le même contexte que Cowie *et al.* (1992) (même table de décroissance de probabilités) en reprenant leurs hypothèses initiales.

9. Cette probabilité est issue de lois thermodynamiques.

converge, plus la probabilité d'accepter des scores plus faibles diminue ; ceci garantit que l'algorithme puisse se terminer.

La configuration avec le score le plus haut est sauvegardée à chaque étape. Cette configuration sera celle qui sera considérée comme résultat indépendamment de la configuration courante à la convergence de l'algorithme.

5. Algorithme global fondé sur des colonies de fourmis

5.1. *Algorithmes à colonies de fourmis*

Les algorithmes à colonies de 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 de résoudre des problèmes complexes. Ce phénomène est appelé intelligence en essaim (Bonabeau et 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 colonies de fourmis.

Le concept principal de l'intelligence en essaim est la stygmergie, c'est-à-dire 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 fondée 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 fondé sur ce paradigme pour le célèbre problème combinatoire du voyageur de commerce (Dorigo et 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 du graphe et offrent des résultats comparables à ceux obtenus par les différentes

méthodes de résolution et heuristiques utilisées en algorithmique des graphes. Leur grand intérêt réside dans leur capacité à s'adapter à un changement de l'environnement. Le lecteur trouvera dans (Dorigo et Stützle, 2004) ou (Monmarche *et al.*, 2009) de bons états de l'art sur la question.

5.2. *ACA : algorithme à colonies de fourmis pour la désambiguïsation lexicale*

Dans cette section, nous présentons notre algorithme à colonies de fourmis pour la désambiguïsation lexicale. Il est nommé dans la suite ACA (pour *Ant Colony Algorithm*).

5.2.1. *Vue d'ensemble*

L'environnement des fourmis est un graphe. Il peut être linguistique – morphologique comme dans (Rouquet *et al.*, 2010) ou morphosyntaxique comme dans (Schwab et Lafourcade, 2007 ; Monmarche *et al.*, 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'est-à-dire un graphe simple (voir figure 1), sans information linguistique externe, afin de mieux faire comprendre la mécanique de nos algorithmes. Dans ce graphe,

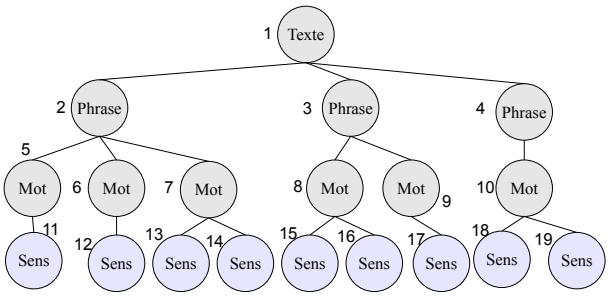


Figure 1. *Environnement utilisé dans cette expérience : texte, phrases et mots correspondent aux nœuds dits normaux (nœuds 1 à 10), un sens de mot correspond à une fourmilière (nœuds 11 à 19).*

nous distinguons deux types de nœuds : les *fourmilières* et les *nœuds normaux*. Suivant les idées développées dans (Schwab, 2005) et (Monmarche *et al.*, 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. Une fourmi a une odeur qui correspond à la définition du sens de leur fourmilière mère. Pour une fourmi, un nœud peut être : (1) *la fourmilière mère* où elle est née ; (2) *une fourmilière ennemie*, toutes les fourmilières qui correspondent 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, la fourmilière 18 est une ennemie car elles ont toutes deux le même père (le nœud 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 3.1.2), de la présence d'énergie, du passage des autres fourmis (les fourmis laissent des traces sur les arcs où elles passent sous la forme de *phéromone*) et du vecteur odeur des nœuds (les fourmis déposent une partie de leur odeur sur les nœuds normaux où elles passent). Une fois arrivée sur la fourmilière d'un autre terme (correspondant à un sens de ce 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. Ainsi ces fourmis pourront aller d'une fourmilière à une autre sans avoir à remonter jusqu'à la racine de l'arbre. 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 donc 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 avec les informations lexicales considérées). Le fait de ne pas considérer dès le départ, dans l'espace de recherche, tous les ponts possibles entre les fourmilières potentiellement amies, mais de laisser leur construction éventuelle aux fourmis cherchant à rejoindre leur fourmilière mère est pertinent dans ce cadre. Notre approche reste constructive tout en limitant le nombre de configurations à explorer[10].

5.2.2. *Détails de l'algorithme*

Les tableaux 1 et 2 présentent les principales notations ainsi que les paramètres (et leurs valeurs) utilisés dans la suite de l'article. Les sections 5.5 et 6.3 reviendront en détail sur les paramètres principaux de l'algorithme et l'estimation des valeurs utilisées dans les expérimentations menées.

5.2.2.1. 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

10. Cette approche se distingue en cela des approches classiques d'optimisation considérant l'ensemble des configurations dans l'espace de recherche initial.

Notation	Description
F_A	Fourmilière correspondant au sens A
f_A	Fourmi née dans la fourmilière F_A
$V(X)$	Vecteur odeur associé à X. X est un nœud ou une fourmi
$E(X)$	Énergie possédée par X. X est un nœud ou une fourmi
$Eval_f(N)$	Évaluation du nœud N par f
$Eval_f(A)$	Évaluation de l'arc A (quantité de phéromone présente) par f
$\varphi_{(t/c)}(A)$	Quantité de phéromone présente sur l'arc A à l'instant t ou au cycle c

Tableau 1. *Principales notations pour l'algorithme à colonies de fourmis*

Notation	Description	Valeur
E_F	Énergie utilisée par une fourmilière pour produire une fourmi	1
E_a	Énergie prise par une fourmi lorsqu'elle arrive sur un nœud	1
θ	Phéromone déposée par une fourmi lors de la traversée d'un arc	1
E_{max}	Énergie maximale que peut porter une fourmi	2-10
δ	Évaporation de la phéromone entre chaque cycle	20-80 %
E_0	Quantité initiale d'énergie sur chaque nœud	5-20
ω	Durée de vie d'une fourmi	5-20 (cycles)
L_V	Longueur du vecteur odeur	30-150 (dim.)
δ_V	Proportion du vecteur odeur déposée par une fourmi lorsqu'elle arrive sur un nœud	10-90 %
c_{ac}	Nombre de cycles de la simulation	100

Tableau 2. *Paramètres et valeurs pour l'algorithme à colonies de fourmis*

et suivant une courbe sigmoïde classiquement utilisée dans les réseaux de neurones artificiels (Lafourcade, 2011). Comme dans (Schwab et Lafourcade, 2007) ou (Guinand et Lafourcade, 2010), notre fonction sigmoïde[11] est $\frac{arctan(x)}{\pi} + \frac{1}{2}$. Ces fourmis se déplacent dans le graphe et cherchent à ramener leur énergie à leur fourmilière mère qui sera alors à même de produire de nouvelles fourmis. Les fourmis ont une durée de vie (ω) limitée (un nombre identique de cycles fixé *a priori* cf. tableau 2). Lorsqu'une fourmi meurt, toute l'énergie qu'elle transportait (et celle qui avait été utilisée par la

11. La fonction sigmoïde a été choisie car elle permet d'avoir une valeur positive même lorsque le niveau d'énergie est négatif. Ainsi, les fourmilières concernées peuvent tout de même fabriquer quelques fourmis. 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 la fourmilière.

fourmilière pour la produire) est déposée sur le nœud où elle se trouve. Ainsi la quantité d'énergie présente dans le système reste constante. C'est un point fondamental qui permet la convergence de l'algorithme vers une solution (renforcement de l'énergie dans quelques fourmilières, au détriment des autres).

5.2.2.2. Phéromone de passage

Les fourmis ont deux types de comportements. 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.

5.2.2.3. Odeur

L'odeur d'une fourmilière est une représentation vectorielle correspondant à la définition du sens correspondant (Schwab *et al.*, 2012). Il s'agit donc de la définition du sens sous forme de vecteurs de nombres entiers. Chaque fourmi née dans cette fourmilière porte la même odeur, le même vecteur (voir section 3.1.2). Les nœuds normaux sont initialisés avec des composantes vides. 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 son passage. Ainsi plus un nœud est proche d'une fourmilière plus il y a de chances 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.section 5.2.2.4). 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.

5.2.2.4. Création et suppression de ponts

Un pont peut être créé lorsqu'une fourmi atteint une fourmilière potentiellement amie, c'est-à-dire 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[12], il y a création d'un pont entre les deux fourmilières. Ce pont est ensuite considéré comme un arc standard par les fourmis, c'est-à-dire que les nœuds qu'il lie sont considérés comme voisins. Si le pont ne porte plus de phéromone, il disparaît.

5.2.3. *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 (en fonction de la durée de vie paramétrée) ; (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 ou retour à la fourmilière ; le changement étant fait de manière probabiliste), la déplacer et 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'est-à-dire 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'est-à-dire 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 ($Eval_f(A) = 1 - \varphi_t(A)$) 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 ($Eval_f(A) = \varphi_t(A)$) et aller vers les nœuds dont l'odeur est proche de la sienne ($Eval_f(N) = \frac{Lesk_{ext}(V(N),V(f_A))}{\sum_{i=1}^{i=k} Lesk_{ext}(V(N_i),V(f_A))}$).

12. De la même manière qu'expliqué en section 5.2.3, sans tenir compte de la phéromone si le pont n'existe pas encore.

5.3. *Exemple illustré*

Grâce au parcours que va réaliser une fourmi (figures 2, 3, 4), nous illustrons maintenant les différentes étapes de l'algorithme ACA sur la phrase « *La souris contrôle l'ordinateur.* ». Les vecteurs des définitions sont représentés sous le nœud correspondant et sont précédés de la lettre V. Le niveau d'énergie est représenté par des barres empilées précédées par la lettre E. Les lignes entre les nœud représentent les chemins et leur épaisseur la concentration de phéromone. Les numéros dans les cercles montrent les déplacements successifs de la fourmi. La quantité d'énergie que la fourmi porte lors de l'étape est matérialisée par les barres empilées à côté du dessin de la fourmi et nommée E(A). Les nombres dans les vecteurs sous les fourmilières correspondent aux mots des définitions des sens correspondants (voir section 3.1.2).

Pour cet exemple, nous suivons une fourmi qui naît dans la fourmilière de ‹*souris/dispositif*› correspondant au premier sens pour le nom ‹*souris*›.

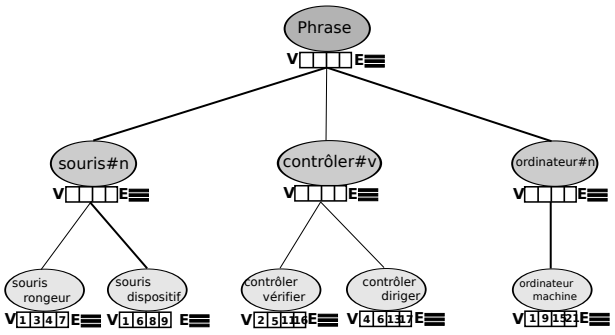


Figure 2. *Exemple : état de l'environnement au départ*

Sur la figure 3, la fourmi débute en mode recherche de nourriture. Elle prend une unité d'énergie sur sa fourmilière (1) et commence à explorer le graphe en démarrant par le nœud correspondant au nom ‹*souris#n*› (2), le seul choix que l'environnement lui offre à ce moment-là. Sur le chemin vers ce nœud, elle dépose sa phéromone puis, en arrivant, elle prend une barre d'énergie. Elle dépose également un peu de son odeur sur le nœud. Cette odeur est matérialisée par le dépôt de deux composants pris au hasard parmi ceux de sa fourmilière mère, soit ceux de ‹*souris/dispositif*›.

La fourmi choisit ensuite de suivre le chemin (3) où elle dépose sa phéromone pour atteindre la racine de l'environnement (4). Elle y dépose également deux composants du vecteur de la fourmilière mère pris au hasard et prend une barre d'énergie. À ce stade, la fourmis possède trois barres d'énergie et une décision pseudo-aléatoire va la faire passer en mode retour à la fourmilière mère.

Comme une fourmi ne peut pas revenir sur un nœud d'où elle vient sauf si c'est sa seule possibilité, notre fourmi a le choix entre aller sur ‹*contrôler#v*› ou sur ‹*ordinateur#n*›.

Les fourmis venant d'‹*ordinateur/machine*› ont déposé une part de leur vecteur sur ‹*ordinateur#n*›. Le vecteur de ce nœud a donc plus d'éléments en commun avec le vecteur de notre fourmis que celui de ‹*contrôler#v*›. Notre fourmi a donc plus de chances d'aller vers ‹*ordinateur#n*›. C'est ce qu'elle fait tout en déposant sa phéromone sur son chemin (5). La fourmi laisse son odeur sur le nœud d'arrivée mais ne prend pas d'énergie puisqu'elle est en mode retour (6). Ensuite, elle ne peut qu'aller (7) vers ‹*ordinateur/machine*› (8).

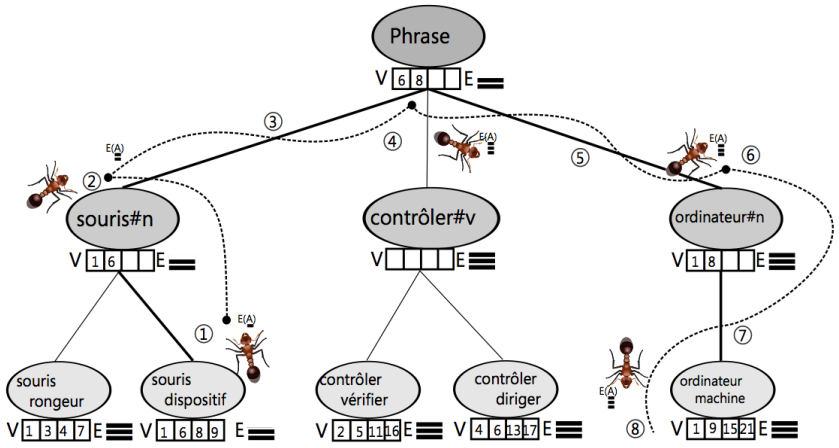


Figure 3. *Exemple : déplacement d'une fourmi, mode recherche (1,2,3,4) et mode retour (5,6,7,8)*

Pour cette fourmi, le nœud ‹*ordinateur/machine*› est un nœud potentiellement ami. Elle a donc la possibilité de construire un pont vers sa fourmilière mère. Ce choix est réalisé de manière pseudo-aléatoire (voir figure 4). Ce pont devient un arc que pourront emprunter d'autres fourmis et en particulier celles de ‹*ordinateur/machine*› et de ‹*souris/dispositif*› donc les vecteurs partagent beaucoup de composants. La fourmi choisit donc de l'emprunter et y dépose sa phéromone (9). Elle atteint sa fourmilière mère et y dépose toute l'énergie qu'elle peut porter (10). La fourmi repasse en mode recherche et reprend ses déplacements jusqu'au nœud où elle va mourir et déposera l'énergie qu'elle transportera alors ainsi que l'énergie qu'il a fallu pour la fabriquer.

5.4. *Évaluation globale, fonction de coût*

À la fin de chaque cycle, nous construisons la configuration courante, c'est-à-dire que, pour chaque mot, nous prenons le sens correspondant à la fourmilière ayant le plus d'énergie.

À la fin de la simulation, le résultat est la configuration qui obtient le plus grand score. Il s'agit d'une modification par rapport à l'algorithme original présenté dans (Schwab *et al.*, 2011).

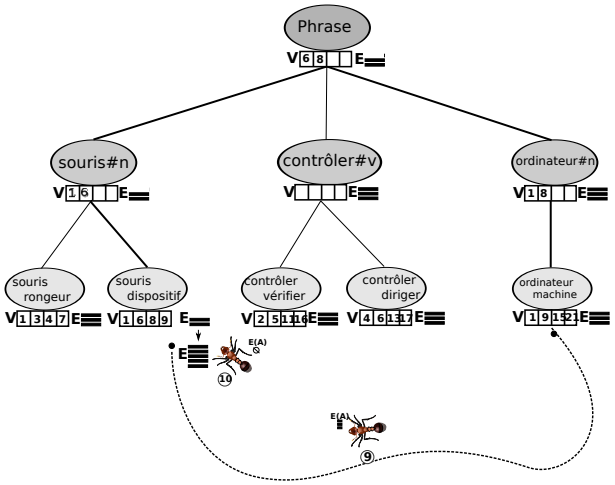


Figure 4. *Exemple : mode retour et création de pont (étapes 9 et 10)*

5.5. *Paramètres importants*

Les paramètres suivants ont une influence sur certains des phénomènes émergents que connaît le système :

– la quantité maximale d'énergie qu'un nœud peut porter (E_{max}) influence la taille du ou des cycles parcourus par une fourmi. Comme les fourmis ne peuvent pas revenir immédiatement sur un arc qu'elles viennent de traverser, elles réalisent forcément un circuit pour revenir à leur fourmilière mère (elles peuvent cependant mourir avant d'y arriver). La taille du circuit dépend du moment où elles décident de revenir, c'est-à-dire de E_{max} ;

– l'évaporation de la phéromone entre chaque cycle du système (δ) est l'une des mémoires du système. Plus il y a d'évaporation, moins les traces des fourmis jouent un rôle important. Les chemins interprétatifs doivent être plus rapidement confirmés par de nouvelles fourmis faute de quoi ils sont plus rapidement oubliés par le système global ;

– la quantité initiale d'énergie sur les nœuds (E_0) et la durée de vie des fourmis (ω) influencent le nombre de fourmis qui peuvent être produites et donc la probabilité de renforcement de certains chemins et circuits plus litigieux pour le système ;

– la taille du vecteur d'odeur (L_v) ainsi que la quantité de composants déposés par une fourmi sur un nœud normal (δ_V) influencent la mémoire globale du système. Plus le vecteur odeur des nœuds normaux est grand, plus longtemps le souvenir du passage d'une fourmi sera conservé. La quantité de composants déposés influence cette mémoire de façon inverse : plus il y en a, moins le souvenir du passage sera long.

Par la dynamique du système, ces paramètres sont liés entre eux. Nous avons mené des expériences afin d'estimer au mieux leurs valeurs. Elles sont décrites section 6.3.

6. Évaluation pratique

Dans cette section, nous présentons tout d'abord notre corpus de test et les métriques utilisées pour évaluer la tâche de désambiguïsation lexicale visée. Nous présentons ensuite les expériences menées et les valeurs des paramètres que nous avons déterminées pour chacun des algorithmes. Enfin nous concluons cette partie par une comparaison des performances de chaque algorithme, à la fois en termes de qualité de la désambiguïsation réalisée et en temps d'exécution.

6.1. *Corpus d'évaluation*

Nous avons testé notre méthode sur le corpus de la tâche *gros grain*, une des dix-huit tâches 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. 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 ; ce qui revient à accepter un intervalle d'erreur. Notre objectif est de tester un système en vue d'une utilisation dans un cadre applicatif réel. L'inventaire de sens grossiers n'étant disponible que pour les 2 269 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*.

Le corpus d'évaluation est composé de cinq textes. Les tableaux 3 et 4 présentent diverses informations sur ce corpus.

Texte	Genre	Nombre de phrases	Nombre de mots	Nombre de mots à annoter	Mots à annoter par phrase
d001	Journalisme	35	951	368	10,51
d002	Critique littéraire	41	987	379	9,24
d003	Voyage	59	1 311	500	8,47
d004	Informatique	76	1 326	677	8,91
d005	Biographies	34	802	345	10,15
Total		245	5 377	2 269	9,26

Tableau 3. *Les cinq textes du corpus*

	Nombre de mots	Nombre de sens	Nombre de sens moyen (tous mots)	Nombre de mots monosémiques	Nombre de sens moyen (mots polysémiques)
d001	368	1 896	5,15	66	6,06
d002	379	2 257	5,96	64	6,96
d003	500	3 234	6,48	91	7,68
d004	677	3 688	5,45	108	6,29
d005	345	2 955	8,57	28	9,23
Total	2 269	14 030	6,18	357	7,15

Tableau 4. *Polysémie dans la version* a posteriori *du corpus*

6.2. *Métriques*

Nous utilisons quatre métriques pour évaluer la qualité des solutions produites. Il s'agit des mesures standard pour l'évaluation des tâches de désambiguïsation lexicale (Navigli, 2009) :

La première métrique, la *Couverture* (C), est définie par le quotient du nombre de réponses produites par le nombre de réponses attendues. En d'autres termes, elle représente la proportion du texte ayant été désambiguïsée :

$$C = \frac{nombre\ de\ réponses\ produites}{nombre\ de\ réponses\ attendues}$$

La deuxième métrique, la *Précision* (P), est définie par le quotient du nombre de réponses correctes produites par le nombre total de réponses produites :

$$P = \frac{sens\ correctement\ annotés}{sens\ annotés}$$

La troisième métrique, le *Rappel* (R), est définie par le quotient du nombre de réponses correctes produites par le nombre total de réponses attendues :

$$R = \frac{sens\ correctement\ annotés}{sens\ à\ annoter}$$

La dernière métrique, la *F-mesure*, F_1, représente la « moyenne harmonique pondérée de la *Précision* et du *Rappel* ». Elle regroupe ainsi P et R dans une seule mesure :

$$F_1 = \frac{2 \cdot P \cdot R}{P\ +\ R}$$

Il est important de noter que, pour les trois algorithmes étudiés ici (hormis l'algorithme exhaustif), la couverture est toujours de 100 %, ce qui veut dire que :

$$P = R = F_1 \text{ puisque } F_1 = \frac{2 \cdot P \cdot P}{P + P} = \frac{2 \cdot P^2}{2 \cdot P} = P$$

6.3. *Tests et configurations expérimentales*

Les objectifs des expériences présentées ici ont été de déterminer les paramètres de chacun des algorithmes en termes de vitesse d'exécution et de qualité des solutions produites, puis de les comparer. Pour chacun des algorithmes, nous avons sélectionné des valeurs de références puis plusieurs expériences de désambiguïsation lexicale ont été conduites avec différentes valeurs de paramètres.

Comme le montre le tableau 5, le recuit simulé nécessite trois paramètres, les algorithmes génétiques et l'algorithme à colonies de fourmis en nécessitent sept[13]. Tester l'ensemble des valeurs pour chaque paramètre est évidement impossible car chacun est dans un domaine continu et même parfois ouvert. En revanche, on peut restreindre ces domaines à un certain nombre de valeurs. Comme ces paramètres sont intercorrélés par construction des algorithmes, il faudrait alors tester l'ensemble des combinaisons, ce qui entraînerait rapidement une explosion combinatoire. L'estimation des valeurs des paramètres peut alors être réalisée de plusieurs manières pouvant être ou non associées :

– *a priori*, avant de tester, en se fondant sur la dynamique de chaque algorithme. Par exemple, pour l'algorithme à fourmis, l'énergie maximale que peut porter une fourmi doit être inférieure à sa durée de vie si on veut qu'elle puisse passer en mode retour et pour l'algorithme génétique, le nombre de chromosomes ne doit pas être trop élevé si on ne veut pas que l'exécution soit trop longue ;

– *empiriquement*, tester manuellement (ou plus exactement semi-manuellement) plusieurs combinaisons de paramètres que l'on pense intéressantes puis analyser quelle est l'influence individuelle des paramètres et quels sont les compromis qu'il s'agit de faire ;

– *par apprentissage automatique*, à l'aide d'une méthode d'optimisation comme, par exemple, un algorithme génétique ou un recuit simulé.

On le voit, l'ensemble de ces techniques nécessite une certaine supervision. Il s'agit même d'une limitation claire de la supervision absolue pour les algorithmes globaux de désambiguïsation lexicale fondée sur des connaissances. Ils ont tous des paramètres qu'il convient de choisir (taille de fenêtre pour *Lesk-contexte* ou les méthodes de Sinha et Mihalcea (2007) et de Mihalcea *et al.* (2004), des paramètres appris automatiquement pour *Degree* (Ponzetto et Navigli, 2010)). Alors que les deux premières méthodes d'estimation des paramètres ont été utilisées dans (Schwab *et al.*, 2011), nous testons ici la troisième, détaillée dans (Tchechmedjiev *et al.*, 2012).

13. Les valeurs de E_F, E_a, et θ sont fixées.

6.4. *Paramètres*

Les paramètres des trois algorithmes sont résumés dans le tableau 5. Pour les fourmis, qui nous intéressent ici, il y a en tout sept paramètres, dont cinq sont discrets (représentés par des entiers) et deux sont continus (représentés par des réels positifs entre zéro et un). Initialement, nous avions déterminé les valeurs de quelques paramètres en appliquant des changements itératifs et indépendants. Bien entendu, une telle approche est limitée du fait qu'il s'agit d'une heuristique s'appuyant sur l'hypothèse que les paramètres du système sont indépendants. Ce n'est pas le cas de l'algorithme à colonies de fourmis comme nous en avons discuté en section 5.5. Vu le nombre de paramètres, même avec une connaissance préalable du fonctionnement de l'algorithme, on peut au mieux choisir des intervalles pour les valeurs des paramètres afin de limiter l'explosion combinatoire.

De plus, du fait du nombre de paramètres et des intervalles de valeurs, une énumération exhaustive n'est pas envisageable. Le calcul du nombre de combinaisons (en prenant des pas de 0,01 pour les paramètres continus) donne : $60 \times 60 \times 100 \times 55 \times 25 \times 35 \times 100 = 17\ 325 \cdot 10^8$ combinaisons ; à cela s'ajoute la nature probabiliste des algorithmes qui requiert au moins 50 à 100 exécutions pour pouvoir calculer si les résultats sont significatifs, on atteint ainsi rapidement $17\ 325 \cdot 10^9$ combinaisons. Même en supposant que l'algorithme n'ait besoin que de 1 seconde pour s'exécuter, la recherche des paramètres prendrait tout de même $549\ 372$ ans.

Pour cette raison, nous nous sommes intéressés à l'application d'une méthode d'estimation automatique pour trouver les paramètres *optimaux* c'est-à-dire un ensemble de valeurs de paramètres qui mène à un F-score aussi haut que possible.

Notre méthode utilise un recuit simulé pour lequel chaque élément de la configuration correspond à un paramètre. En utilisant le premier texte pour l'apprentissage, les valeurs obtenues après ce processus détaillé dans (Tchechmedjiev *et al.*, 2012) sont $\omega = 10$, $E_a = 1$, $E_{max} = 8$, $E_0 = 10$, $\delta_v = 0,9$, $\delta = 0,9$, $L_V = 90$.

6.5. *Comparaison qualitative*

Nous analysons nos résultats en calculant s'ils sont significatifs par une analyse de variance (ANOVA) (Miller, 1997). Il faut donc vérifier l'hypothèse de normalité. Nous avons ainsi calculé la corrélation entre les quantiles théoriques de la distribution normale et les quantiles empiriques. Pour toutes les mesures et tous les algorithmes, nous avions toujours une corrélation d'au moins 0,99, par conséquent le test de Shapiro et Wilk (1965) est non significatif. En revanche, le test d'homoscédasticité de Levene (1960) est significatif avec p toujours inférieur à 10^{-6}.

Le tableau 6 présente les résultats en fonction du score F_1, du temps d'exécution et du nombre d'évaluations de la mesure de similarité, ainsi que les écarts-types respectifs pour les texte 2 à 5. Les résultats sont significatifs deux à deux pour les trois algorithmes avec p-ajusté inférieur à 0,01. Sachant que le premier texte à été utilisé

CR	MR	MN	λ	STH
1,0	**0,15**	1	1 000	10
0,9	0,3	20	**500**	**20**
0,6	0,8	50	200	30
0,3	1,0	**80**	50	40

(a) Algorithme génétique

T_0	ClR	IN
1 000	1,00	2 000
700	**0,9**	**1 000**
400	0,5	700
100	0,3	400
1	0,1	100

(b) Recuit simulé

ω	E_{max}	E_0	L_V	δ	δ_V
20	10	**20**	150	**80**	**10**
15	**8**	15	120	60	30
10	6	10	**90**	40	50
5	4	5	60	20	70
	2		30	90	

(c) Colonies de fourmis

Tableau 5. *Valeurs des paramètres testés (valeurs optimales en **gras**)*

Algorithme	F_1 (%)	σ_{F_1}	$Time(s)$	σ_{Time}	Sim. Eval.	$\sigma(S.Ev.)$
1er sens	77,59	N/A	N/A	N/A	N/A	N/A
A.C.A.	76,41	0,0048	65,46†	0,199	1 559 049†	17 482,45
S.A.	74,23†	0,0028	1 436,6†	167,3	4 405 304†	50 805,27
G.A.	73,98†	0,0052	4 537,6†	963,2	137 158 739†	13 784,43

Tableau 6. *Comparaisons des scores F_1, temps d'exécution et nombre d'évaluation de la mesure de similarité pour les algorithmes ($\dagger \leftrightarrow p < 0,01$) sur les textes 2 à 5*

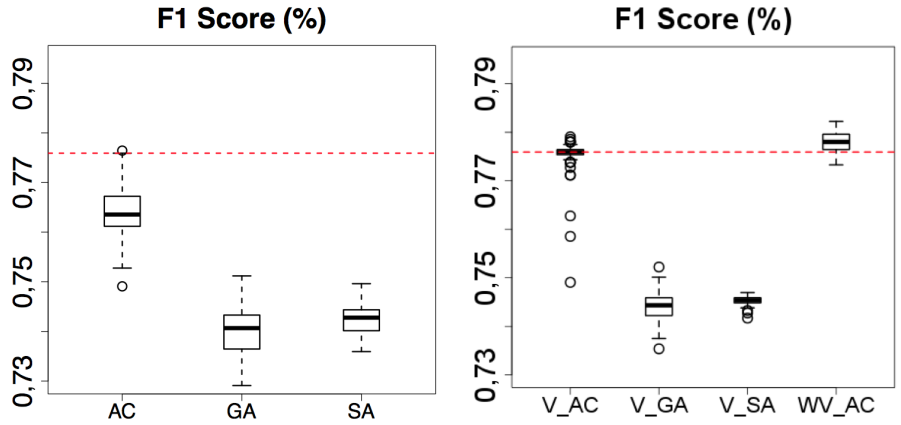
pour la recherche de paramètres, les scores F_1 présentés ici sont calculés sur les quatre autres textes seulement, afin de supprimer tout biais.

De même, la tableau 7 présente les scores F_1 pour les trois algorithmes avec plusieurs stratégies de fusion tardive : un vote majoritaire pour les trois algorithmes ainsi qu'une stratégie de vote par majorité pondéré pour ACA. Nous appliquons également

Algorithme	F_1 (%)	σ_{F_1} (%)
Vote majoritaire pondéré ACA	77,79†	0,18
Référence S,P,F,	77,59	N/A
Vote majoritaire ACA	77,50†	0,48
Vote majoritaire SA	75,18†	0,10
Vote majoritaire GA	74,53†	0,31

Tableau 7. *Comparaison des scores F_1 après l'application d'une stratégie de vote ($\dagger \leftrightarrow p < 0,01$) sur les textes 2 à 5*

un test *post hoc* de Tukey HSD (Abdi et Williams, 2010) qui indique la significativité deux à deux entre tous les algorithmes et types de votes avec un p-ajusté inférieur à $0,01$.



(a) Exécution normale **(b)** Vote par majorité

Figure 5. *Boîtes à moustaches des scores F_1 comparés à la référence du premier sens (pointillé)*

Les figures 5a et 5b présentent respectivement des boîtes à moustaches des distributions des scores F_1 pour les trois algorithmes, avec et sans vote, par rapport à la référence du sens le plus courant.

Vis-à-vis du score F_1, SA et GA obtiennent des résultats similaires, même si les résultats de SA sont légèrement meilleurs et présentent une variabilité de la distribution de scores inférieure.

Pour ACA, les scores sont en moyenne meilleurs de 1,61 % par rapport au SA et 1,76 % meilleur que le GA, avec des variabilités similaires. Les trois algorithmes

obtiennent tous des résultats inférieurs à la référence du sens le plus fréquent, même si le maximum pour ACA s'en rapproche. Après l'application du vote par majorité sur les résultats dix par dix, pour SA et GA il y a une légère amélioration des scores, respectivement une augmentation de 0,95 % et de 0,61 % avec $p < 0,01$. Pour ACA, il y a une amélioration plus importante ($p < 0,01$) de 1,09 % (malgré les quelques données aberrantes représentées par des cercles). ACA a tendance à converger sur des solutions proches de la référence du premier sens. Après le vote, la distribution est presque centrée autour de la référence du premier sens en ce qui concerne le score. Quand on ajoute au vote des poids donnés par les valeurs d'énergie de l'algorithme à colonies de fourmis, le score moyen augmente de 0,29 %, ce qui le place 0,20 % au-dessus de la référence du sens le plus fréquent. De plus, la distribution est bien plus compacte et ne comporte plus de valeurs aberrantes. Cette stratégie ne peut pas être utilisée avec les deux autres algorithmes qui ne pondèrent pas les sens d'un même mot.

En ce qui concerne les temps d'exécution, il y a d'énormes différences entre les algorithmes, le plus lent étant l'algorithme génétique, qui, en moyenne prend 1 h 30 ($\pm$ 16 min). Le recuit simulé est bien plus rapide et prend en moyenne 24 min ($\pm$ 2,8 min), mais reste tout de même bien plus lent que l'algorithme à colonies de fourmis qui, en moyenne, converge en 65 s ($\pm$ 190 ms). Comme l'on pourrait s'y attendre, le nombre d'évaluations de la mesure de similarité est directement corrélé avec les temps d'exécution des algorithmes (cor_p = 0,9969).

6.6. *Comparaison avec les autres systèmes sur cette tâche*

Notre article concerne les algorithmes globaux et principalement trois algorithmes que nous comparons ensemble. Une section sur la comparaison avec les autres systèmes sur cette tâche devrait donc fortement surprendre le lecteur attentif. Cependant, nous l'incluons tout de même car elle permet de poser certaines perspectives de nos travaux actuels et futurs.

Nous nous sommes limités ici aux systèmes qui désambiguïsent l'ensemble du texte et non pas seulement un sous-ensemble, afin de permettre une comparaison équitable. En outre, puisque les résultats sont pour les cinq textes, nous examinons également les résultats de l'algorithme à fourmis pour l'ensemble du corpus contrairement à la section précédente. Il s'agit clairement d'un biais favorable (mais relativement faible) en faveur des fourmis. Par rapport aux autres participants d'origine, nos algorithmes sont en avance sur tous les autres systèmes non supervisés et battent le plus faible système supervisé. La *baseline* du premier sens qui est également une méthode supervisée, rappelons-le, est également battue ce qui est très rare.

Nous avons ajouté le récent *SEL + 100* (Miller *et al.*, 2012) qui est le premier, à notre connaissance, à nous battre alors qu'il fait partie des systèmes non supervisés. Son algorithme local est fondé sur une extension des définitions et nous prévoyons de le tester prochainement avec notre algorithme à colonies de fourmis.

Une autre amélioration possible de l'algorithme local peut être étudiée du côté de BabelNet avec lequel Navigli et Ponzetto (2012) obtiennent pour l'algorithme *Degree* un score presque aussi bon que l'ACA (– 0,63 % avec l'algorithme simple et – 1,75 % avec le vote). Cependant, il est important de noter qu'après une analyse des scores par partie du discours, *Degree* présente des résultats sensiblement plus élevés pour les noms (85 % contre 76,35 %), alors que l'ACA est plus performant pour les adjectifs, les adverbes et les verbes (83,98 %, 82,44 %, 74,16 %).

System	A	P	R	F_1
UoR-SSI‡	100	83,21	83,21	83,21
NUS-PT‡	100	82,50	82,50	82,50
NUS-ML‡	100	81,58	81,58	81,58
LCC-WSD‡	100	81,45	81,45	81,45
SEL+100	100	81,03	81,03	81,03
GPLSI‡	100	79,55	79,55	79,55
ACA (vote pondéré)	100	79,03	79,03	79,03
First Sense Baseline	100	78,89	78,89	78,89
ACA (vote)	100	78,76	78,76	78,76
UPV-WSD‡	100	78,63	78,63	78,63
ACA (WN)	100	77,64	77,64	77,64
Degree (BabelNet)	100	77,01	77,01	77,01
Recuit simulé (vote)	100	75,18	75,18	75,18
Algorithmes génétiques (vote)	100	74,53	74,53	74,53
Recuit simulé	100	74,23	74,23	74,23
Algorithmes génétiques	100	73,98	73,98	73,98
Page Rank (BabelNet)	100	72,60	72,60	72,60
TKB-UO	100	70,21	70,21	70,21
RACAI-SYNWSD	100	65,71	65,70	65,70

Tableau 8. *Comparaison des systèmes ayant 100 % de couverture sur le corpus de SemEval2007, tâche 7 ; ‡ ↔ systèmes supervisés ; les systèmes en italique sont postérieurs à la campagne de 2007 ; les systèmes en* ***gras*** *sont ceux présentés dans cet article.*

7. Conclusions et perspectives

Dans cet article, nous avons confronté un algorithme à colonies de fourmis pour la désambiguïsation lexicale à deux autres algorithmes stochastiques issus de l'état de l'art (un algorithme génétique et un recuit simulé). Ces trois algorithmes sont des algorithmes dits globaux qui utilisent une mesure locale non supervisée dont ils propagent les résultats au niveau du texte pour désambiguïser l'ensemble des mots. En termes d'efficacité temporelle, les algorithmes à colonies de fourmis sont plus rapides d'un ordre de dix par rapport au recuit simulé, lui-même plus rapide d'un ordre de dix par rapport aux algorithmes génétiques. Permettant, de plus, un parallélisme important, contrairement au recuit simulé, les algorithmes à colonies de fourmis en sont d'autant plus efficaces. Cette efficacité a par ailleurs rendu possible l'ajout d'une légère supervision dans l'algorithme pour automatiser la recherche des paramètres. Il en résulte une amélioration sensible des résultats. Avec comme algorithme local $Lesk_{ext}$ et en utilisant une stratégie de vote pondéré, notre algorithme à colonies de fourmis bat maintenant la *baseline* du premier sens sur la tâche considérée dans cet article, *baseline* que seulement un autre système, (Miller *et al.*, 2012), dépasse à notre connaissance.

Il serait intéressant de pouvoir interpréter les résultats afin de mieux cerner le comportement de nos algorithmes (influence du contexte utilisé, influence du genre du texte à désambiguïser, catégorisation des ambiguïtés résolues ou non résolues notamment en distinguant *a posteriori* les mots utilisés dans leur sens le plus fréquent des autres mots). Toutefois le corpus utilisé est relativement peu conséquent pour qu'on puisse tirer des généralités d'une étude linguistique des résultats obtenus par cette seule expérience. D'autre part, ces points sont relativement peu mis en avant dans les campagnes d'évaluation fournissant ces corpus. Nous manquons ainsi d'informations sur les textes eux-mêmes et les raisons pour lesquelles ils ont été choisis, rendant nécessaire, afin d'interpréter les résultats, une étude linguistique approfondie des textes et la caractérisation des phénomènes qu'ils contiennent[14]. En revanche les résultats de cette expérience ouvrent des perspectives de recherche intéressantes plus facilement abordables dans un premier temps. Nous cherchons, par exemple, à mieux comprendre l'influence des différents paramètres afin de savoir s'ils doivent être adaptés, par exemple, au genre du texte, au score local utilisé, à l'environnement utilisé (graphe issu d'une analyse syntaxique, morphologique...) ou bien encore à leur nature différente (oral transcrit, reconnaissance de la parole, par exemple). Une autre piste importante concerne les possibilités de fusion d'informations provenant de sources différentes (WordNet, Wikipédia, Wiktionnaire...), de langues différentes (définitions alignées permettant, par exemple, d'utiliser uniquement des définitions italiennes pour désambiguïser un terme anglais). Cette fusion peut être : (1) précoce, c'est-à-dire avant la désambiguïsation, en concaténant des informations comme le font Ponzetto et Na-

14. Nous avons toutefois amorcé un travail dans ce sens pour les algorithmes à colonies de fourmis en considérant conjointement le corpus utilisé dans cet article et celui de la campagne récente SemEval 2013 à laquelle nous avons participé.

vigli (2010) avec WordNet et Wikipédia ; (2) tardive, après la désambiguïsation par une stratégie de vote, par exemple ; (3) hybride. Ce dernier type de fusion d'informations pourra être possible par l'utilisation de castes comme le fait Schwab (2005) avec des fonctions lexicales. Dans cette hypothèse, lors des retours à leur fourmilière mère, certaines fourmis utilisent alors les informations issues d'une certaine source, les autres castes utilisant d'autres sources.

8. Bibliographie

Abdi H., Williams L. J., *Tukey's Honestly Signifcant Difference (HSD) Test*, Sage, 2010.

Agirre E., Edmonds P., *Word Sense Disambiguation : Algorithms and Applications (Text, Speech and LT)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006.

Banerjee S., Pedersen T., « An adapted Lesk algorithm for word sense disambiguation using WordNet », *CICLing 2002*, Mexico City, February, 2002.

Bonabeau É., Théraulaz G., « L'intelligence en essaim », *Pour la science*, nº 271, p. 66-73, 2000.

Brin S., Page L., « The anatomy of a large-scale hypertextual Web search engine », *Proceedings of the seventh international conference on World Wide Web 7*, WWW7, Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, p. 107-117, 1998.

Brody S., Lapata M., « Good Neighbors Make Good Senses : Exploiting Distributional Similarity for Unsupervised WSD », *Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)*, Manchester, UK, p. 65-72, 2008.

Cowie J., Guthrie J., Guthrie L., « Lexical disambiguation using simulated annealing », *COLING 1992*, vol. 1, Nantes, France, p. 359-365, août, 1992.

Cramer I., Wandmacher T., Waltinger U., *WordNet : An electronic lexical database*, Springer, chapter Modeling, Learning and Processing of Text Technological Data Structures, 2010.

Deerwester S. C., Dumais S. T., Landauer T. K., Furnas G. W., Harshman R. A., « Indexing by Latent Semantic Analysis », *Journal of the American Society of Information Science*, 1990.

Deneubourg J.-L., Gross S., Franks N., Pasteels J.-M., « The blind leading the blind : Modeling chemically mediated army ant raid patterns », *Journal of Insect Behavior*, vol. 2, p. 719-725, 1989.

Dorigo M., Gambardella L., « Ant colony system : A cooperative learning approach to the traveling salesman problem », *IEEE Transactions on Evolutionary Computation*, vol. 1, p. 53-66, 1997.

Dorigo, Stützle, *Ant Colony Optimization*, MIT-Press, 2004.

Drogoul A., « When ants play chess (or can strategies emerge from tactical behaviors) », *Maamaw'1993*, 1993.

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

Gale W., Church K., Yarowsky D., « One sense per discourse », *Fifth DARPA Speech and Natural Language Workshop*, Harriman, New-York, États-Unis, p. 233-237, February, 1992.

Gelbukh A., Sidorov G., Han S. Y., « Evolutionary Approach to Natural Language WSD through Global Coherence Optimization », *WSEAS Transactions on Communications*, vol. 2, n° 1, p. 11-19, 2003.

Guinand F., Lafourcade M., *Artificial Ants. From Collective Intelligence to Real-life Optimization and Beyond*, Lavoisier, chapter 20 - Artificial ants for NLP, p. 455-492, 2010.

Hirst G., St-Onge D. D., « Lexical chains as representations of context for the detection and correction of malapropisms », *WordNet : An electronic Lexical Database. C. Fellbaum. Ed. MIT Press. Cambridge. MA*p. 305-332, 1998. Ed. MIT Press.

Ide N., Véronis J., « WSD : the state of the art », *Computational Linguistics*, vol. 28, n° 1, p. 1-41, 1998.

Lafourcade M., Lexique et analyse sémantique de textes – structures, acquisitions, calculs, et jeux de mots, HDR, Université Montpellier II, décembre, 2011.

Leacock C., Chodorow M., « Combining local context and WordNet similarity for word sense identification », *WordNet : An electronic Lexical Database. C. Fellbaum. Ed. MIT Press. Cambridge. MA*, 1998.

Lesk M., « Automatic sense disambiguation using MRD : how to tell a pine cone from an ice cream cone », *Proceedings of SIGDOC '86*, ACM, New York, NY, USA, p. 24-26, 1986.

Levene H., « Robust tests for equality of variances », *in* I. Olkin (ed.), *Contributions to probability and statistics*, Stanford Univ. Press., Palo Alto, CA, 1960.

Li Y., Bandar Z. A., McLean D., « An Approach for Measuring Semantic Similarity between Words Using Multiple Information Sources », *IEEE Trans. on Knowl. and Data Eng.*, vol. 15, n° 4, p. 871-882, July, 2003.

Lin D., « An Information-Theoretic Definition of Similarity », *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML'98, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 296-304, 1998.

Mihalcea R., Tarau P., Figa E., « PageRank on semantic networks, with application to word sense disambiguation », *Proceedings of the 20th international conference on Computational Linguistics*, COLING'04, Association for Computational Linguistics, Stroudsburg, PA, USA, 2004.

Miller R. G., *Beyond ANOVA : Basics of Applied Statistics (Texts in Statistical Science Series)*, Chapman & Hall/CRC, January, 1997.

Miller T., Biemann C., Zesch T., Gurevych I., « Using Distributional Similarity for Lexical Expansion in Knowledge-based Word Sense Disambiguation », *Proceedings of COLING 2012*, The COLING 2012 Organizing Committee, Mumbai, India, p. 1781-1796, December, 2012.

Monmarche N., Guinand F., Siarry P. (eds), *Fourmis artificielles et traitement de la langue naturelle*, Lavoisier, Prague, Czech Republic, 2009.

Navigli R., « WSD : a Survey », *ACM Computing Surveys*, vol. 41, n° 2, p. 1-69, 2009.

Navigli R., Lapata M., « An Experimental Study of Graph Connectivity for Unsupervised Word Sense Disambiguation », *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, p. 678-692, April, 2010.

Navigli R., Litkowski K. C., Hargraves O., « SemEval-2007 Task 07 : Coarse-Grained English All-Words Task », *SemEval-2007*, Prague, Czech Republic, p. 30-35, June, 2007.

Navigli R., Ponzetto S. P., « BabelNet : The automatic construction, evaluation and application of a wide-coverage multilingual semantic network », 2012. www.dx.doi.org/10.1016/j.artint.2012.07.004.

Pedersen T., Banerjee S., Patwardhan S., Maximizing Semantic Relatedness to Perform WSD, Research report, University of Minnesota Supercomputing Institute, March, 2005.

Pirró G., Euzenat J., « A Feature and Information Theoretic Framework for Semantic Similarity and Relatedness », *in* P. Patel-Schneider, Y. Pan, P. Hitzler, P. Mika, L. Zhang, J. Pan, I. Horrocks, B. Glimm (eds), *The Semantic Web - ISWC 2010*, vol. 6496 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, p. 615-630, 2010.

Ponzetto S. P., Navigli R., « Knowledge-rich Word Sense Disambiguation rivaling supervised systems », *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, p. 1522-1531, 2010.

Rada R., Mili H., Bicknell E., Blettner M., « Development and application of a metric on semantic nets », *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 19, n° 1, p. 17-30, January, 1989.

Resnik P., « Using information content to evaluate semantic similarity in a taxonomy », *Proceedings of the 14th international joint conference on Artificial intelligence - Volume 1*, IJCAI'95, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, p. 448-453, 1995.

Rouquet D., Falaise A., Schwab D., Boitet C., Bellynck V., Nguyen H.-T., Mangeot M., Guilbaud J.-P., Rapport final de synthèse, passage à l'échelle et implémentation : extraction de contenu sémantique dans des masses de données textuelles multilingues, Technical report, Agence nationale de la recherche, décembre, 2010.

Schwab D., Approche hybride pour la modélisation, la détection et l'exploitation des fonctions lexicales en vue de l'analyse sémantique de texte., PhD thesis, Université Montpellier II, 2005.

Schwab D., Goulian J., Guillaume N., « Désambiguïsation lexicale par propagation de mesures sémantiques locales par algorithmes à colonies de fourmis », *Traitement Automatique des Langues Naturelles (TALN)*, Montpellier, France, 2011.

Schwab D., Goulian J., Tchechmedjiev A., Blanchon H., « Ant Colony Algorithm for the Unsupervised Word Sense Disambiguation of Texts : Comparison and Evaluation », *Proceedings of COLING 2012*, The COLING 2012 Organizing Committee, Mumbai, India, p. 2389-2404, December, 2012.

Schwab D., Lafourcade M., « Lexical Functions for Ants Based Semantic Analysis », *ICAI'07-The 2007 International Conference on Artificial Intelligence*, Las Vegas, Nevada, USA, juin, 2007.

Seco N., Veale T., Hayes J., « An Intrinsic Information Content Metric for Semantic Similarity in WordNet », *Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain*, p. 1089-1090, 2004.

Shapiro S. S., Wilk M. B., « An Analysis of Variance Test for Normality (Complete Samples) », *Biometrika*, vol. 52, n° 3/4, p. 591-611, Dec., 1965.

Silber H. G., McCoy K. F., « Efficient text summarization using lexical chains », *Proceedings of the 5th international conference on Intelligent user interfaces*, IUI '00, ACM, New York, NY, USA, p. 252-255, 2000.

Sinha R., Mihalcea R., « Unsupervised Graph-basedWord Sense Disambiguation Using Measures of Word Semantic Similarity », *Proceedings of the International Conference on Semantic Computing*, ICSC '07, IEEE Computer Society, Washington, DC, USA, p. 363-369, 2007.

Tchechmedjiev A., Goulian J., Schwab D., Sérasset G., « Parameter estimation under uncertainty with Simulated Annealing applied to an ant colony based probabilistic WSD algorithm », *Proceedings of the First International Workshop on Optimization Techniques for Human Language Technology*, The COLING 2012 Organizing Committee, Mumbai, India, p. 109-124, December, 2012.

Vasilescu F., Langlais P., Lapalme G., « Evaluating variants of the Lesk approach for disambiguating words », *Proceedings of LREC 2004*, Lisbon, Portugal, p. 633-636, May, 2004.

Wu Z., Palmer M., « Verbs semantics and lexical selection », *Proceedings of the 32nd annual meeting on Association for Computational Linguistics*, ACL '94, Association for Computational Linguistics, Stroudsburg, PA, USA, p. 133-138, 1994.

A. Algorithmes

A.1. *Algorithmes généraux*

La fonction *ScoreConfig* renvoie un score entre deux sens (score local). S'il s'agit plus particulièrement de la fonction $Lesk_{ext}$ présentée en section 3.1.2, elle peut également renvoyer le score entre deux vecteurs de définition.

Algorithme 1 *ScoreConfig*

Entrée : C, la configuration dont on veut le score
Sortie : Le score de la configuration C

$score \leftarrow 0$
pour $i = 1..(N-1)$ **faire**
  pour $j = (i+1)..N$ **faire**
    $score \leftarrow score + Score(C[i], C[j])$
  fin pour
fin pour

A.2. *Recuit simulé*

Algorithme 2 Recuit simulé

$Initialiser()$
tant que $Evaluer()$ **faire**
  $Recuit()$
fin tant que

La fonction $Rand(w_i)$ donne un sens au hasard parmi les sens possibles pour le $i$ème mot du texte.

Algorithme 3 *Initialiser* : génération de la configuration initiale

pour $i = 1..N$ **faire**
  $C_{courante}[i] \leftarrow Rand(w_i)$
fin pour
$s \leftarrow ScoreConfig(C_{courante})$
$C_{meilleure} \leftarrow C_{courante}$
$s_{meilleur} \leftarrow s$

Algorithme 4 *Evaluer* : évaluation de la configuration à la fin du cycle courant

si on est dans le premier cycle **alors**
  return VRAI
fin si
tant que $i <= N$ **et** $C_{courante}(i) = C_{précédente}(i)$ **faire**
  $i \leftarrow i + 1$
fin tant que
si $i <= N$ **alors**
  $C_{précédente} \leftarrow C_{courante}$
  $TC \leftarrow TC \cdot ClR$
  return true
sinon
  return false
fin si

La fonction $Copie(C)$ réalise une copie de la configuration C. La fonction $Rand(X, Y)$ renvoie un nombre entier au hasard entre X inclus et Y non inclus.

Algorithme 5 *Recuit* : itérations de recuit

pour $i = 1..IN$ **faire**
  $C'_{courante} \leftarrow Copie(C_{courante})$
  $r \leftarrow Rand(0, N)$
  $C'_{courante}(r) \leftarrow Rand(w_r)$
  $s' \leftarrow ScoreConfig(C'_{courante})$, $\Delta E \leftarrow s - s'$
  si $\Delta E < 0$ **alors**
    $C_{courante} \leftarrow C'_{courante}$
    $s \leftarrow s'$
    si $s_{meilleure} < s$ **alors**
      $C_{meilleure} \leftarrow C_{courante}$
      $s_{meilleure} \leftarrow s$
    fin si
  sinon
    $sel \leftarrow Rand(0, 100)$
    si $sel < 100 \cdot e^{\frac{-\Delta E}{T}}$ **alors**
      $C_{courante} \leftarrow C'_{courante}$
      $s \leftarrow s'$
      si $s_{meilleure} < s$ **alors**
        $C_{meilleure} \leftarrow C_{courante}$
        $s_{meilleure} \leftarrow s$
      fin si
    fin si
  fin si
fin pour

A.3. *Algorithme génétique*

Rappelons qu'un individu correspond ici à une configuration. Les fonctions utilisables sur les configurations sont donc utilisables sur les individus (*ScoreConfig* par exemple).

Algorithme 6 Algorithme génétique

InitPopulation()
$scoreIdentique \leftarrow 0$
$precedentScore \leftarrow 0$
tant que $scoreIdentique \leq STH$ **faire**
  $P_n = P_{n+1}$
  $Selection()$
  $Reproduction()$
  $Mutation()$
  si $meilleurScore = precedentScore$ **alors**
    $scoreIdentique \leftarrow scoreIdentique + 1$
  sinon
    $scoreIdentique \leftarrow 0$
    $scoreIdentique \leftarrow precedentScore$
  fin si
fin tant que

Algorithme 7 *InitPopulation*

$P_n = \{\}$
pour $i = 1..\lambda$ **faire**
  pour $j = 1..N$ **faire**
    $P_{n+1}(i,j) \leftarrow Rand(w_i)$
  fin pour
  $scores[i] \leftarrow ScoreConfig(P_{n+1}(i))$
fin pour

La fonction $Tri(P, T)$ ordonne dans l'ordre croissant la population P en fonction des scores du tableau T.

La fonction $Pop(P)$ enlève et renvoie le premier individu de la population ordonnée dans l'ordre croissant P en fonction des scores du tableau T.

Algorithme 8 *Selection* : algorithme de sélection

pour $i = 1..\lambda$ **faire**
  $scores[i] \leftarrow ScoreConfig(P_n(i))$
fin pour
$Tri(P_{croisement}, scores)$
$\{meilleur, meilleurScore\} \leftarrow Pop(P_n, scores)$
$meilleurScore \leftarrow scores[0]$
si $S_{C_{meilleur}} < meilleurScore$ **alors**
  $C_{meilleur} \leftarrow meilleur, S_{C_{meilleur}} \leftarrow meilleurScore$
  $SC \leftarrow 0$
sinon si $S_{C_b} = meilleurScore$ **alors**
  $SC \leftarrow SC + 1$
fin si
$courant \leftarrow meilleur$
si $scoreCourant = meilleurScore$ **alors**
  $scoreIdentique \leftarrow scoreIdentique + 1$
fin si
$scoreCourant \leftarrow meilleurScore$
tant que non $Vide(P_n)$ **faire**
  $impactRatio \leftarrow \frac{scoreCourant}{meilleurScore}$
  $sélection \leftarrow Rand(0, 100)$
  si $100 \cdot impactRatio \cdot CR > select$ **alors**
    $Ajoute(P_{croisement}, courant)$
  sinon
    $Ajoute(P_{n+1}, courant)$
  fin si
  $\{courant, scoreCourant\} \leftarrow Pop(P_n, scores)$
fin tant que
$Ajoute(P_{n+1}, meilleur)$

Algorithme 9 *Reproduction* : algorithme de croisement

si $|P_{croisement}|$ **mod** $2 = 1$ **alors**
  $Ajoute(P_{n+1}, Pop(P_{croisement}))$
fin si
tant que $|P_{croisement}| > 0$ **faire**
  $parentA \leftarrow Pop(P_{croisement})$
  $parentB \leftarrow Pop(P_{croisement})$
  $pivot1 \leftarrow Rand(1, N)$
  $pivot2 \leftarrow Rand(pivot1 + 1, N)$
  $descendanceA \leftarrow \{0\}$
  $descendanceB \leftarrow \{0\}$
  pour i **in** $1..N$ **faire**
    si $i > pivot1$ **et** $i < pivot2$ **alors**
      $descendanceA_i \leftarrow parentB_i$
      $descendanceB_i \leftarrow parentA_i$
    sinon
      $descendanceA_i \leftarrow parentA_i$
      $descendanceB_i \leftarrow parentB_i$
    fin si
  fin pour
  $Ajoute(P_{n+1}, descendanceA)$
  $Ajoute(P_{n+1}, descendanceB)$
fin tant que

Algorithme 10 *Mutation* : algorithme de mutation

pour i **in** $1..\lambda$ **faire**
  $select \leftarrow Rand(0, 100)$
  si $100 \cdot MR > select$ **alors**
    pour j **in**$1..MN$ **faire**
      $allele \leftarrow Rand(1, N)$
      $P_{n+1}(i, allele) \leftarrow Rand(w_{i,allele})$
    fin pour
  fin si
fin pour

A.4. *Algorithme à colonies de fourmis*

Algorithme 11 Algorithme à colonies de fourmis

pour i **in** $1..c_{ac}$ **faire**
  $traiteFourmilières()$
  $déplaceFourmis()$
  $traiteArcs()$
  $traiteFourmis()$
fin pour

Chaque fourmilière F_i, produit une fourmi f_{F_n} de façon probabiliste suivant une fonction sigmoïde classiquement utilisée dans les réseaux de neurones artificiels (Lafourcade, 2011). Comme dans (Schwab et Lafourcade, 2007) ou (Guinand et Lafourcade, 2010), notre fonction sigmoïde est $\frac{arctan(x)}{\pi} + \frac{1}{2}$.

Algorithme 12 *traiteFourmilières*

Entrée : F, l'ensemble des fourmilières avec $|F| = n$, P_{c-1}, la population de fourmis à la fin du cycle précédant
Sortie : P_c, la population des fourmis qui auront à se déplacer au cours du cycle courant

$P_c \leftarrow P_{c-1}$
pour i **in** $1..n$ **faire**
  si $Rand(0,1) < Sygmo(E(F_n))$ **alors**
    $P_c \leftarrow P_c \cup f_{F_n}$
  fin si
fin pour

Algorithme 13 *déplaceFourmis*

Entrée : P_c, la population des fourmis au cycle courant avec $|P_c| = n$

pour i **in** $1..n$ **faire**
  si $Rand(0,1) < \frac{E(f)}{E_{max}}$ **alors**
    $mode(f_i) \leftarrow chercheFourmilère$
  fin si
  si $mode(f_i) = chercheÉnergie$ **alors**
    $choisirDestinationÉnergie(f_i)$
  sinon
    $choisirDestinationFourmilèreMère(f_i)$
  fin si
fin pour

La fonction $Origine(f)$ renvoie le nœud fourmilière d'origine de la fourmi, la procédure $creerPont(N_1, N_2)$ crée un arc entre les nœuds fourmilières N_1 et N_2 et la fonction *estNœud*(N) renvoie vrai si N est un nid.

Algorithme 14 *choisirDestinationÉnergie*

Entrée : f, une fourmi. N, le nœud où se trouve f

pour i **in** $voisins(N)$ **faire**
  $poids(i) \leftarrow \frac{1+E(i)}{1+\varphi(Arc(N,i))}$
fin pour
si $N \neq Origine(f)$ **et** $estNid(N)$ **alors**
  $poids(N) = 1 + E(Origine(f))$
fin si

$$poidsTotal \leftarrow \sum_{p \in poids} p$$

$cumul \leftarrow 0$
pour i **in** $voisins(N) \cup \{N\}$ **et** $cumul \geq Rand(0,1)$ **faire**
  $cumul \leftarrow cumul + \frac{poids(i)}{poidsTotal}$
fin pour
si $i = N$ **alors**
  $creerPont(N, Origine(f))$
  $emprunterArc(f, N, Arc(N, Origine(i)))$
sinon
  $emprunterArc(f, N, Arc(N, i))$
fin si

La fonction $score(V1, V2)$ donne le score *Lesk* entre le vecteur V1 et le vecteur V2.

Algorithme 15 *choisirDestinationFourmilièreMère*

Entrée : f, une fourmi. N, le nœud où se trouve f

pour i **in** $voisins(N)$ **faire**
  $poids(i) \leftarrow \frac{Score(V(N),V(i)))}{1+\varphi(Arc(N,i))}$
fin pour
si $N \neq Origine(f)$ **et** $Nid(N)$ **alors**
  $poids(Origine(f)) = Score(V(N), V(i))))$
fin si

$$poidsTotal \leftarrow \sum_{p \in poids} p$$

$cumul \leftarrow 0$
pour i **in** $voisins(N) \cup \{Origine(f)\}$ **et** $cumul \geq Rand(0, 1)$ **faire**
  $cumul \leftarrow cumul + \frac{poids(i)}{poidsTotal}$
fin pour
si $i = N$ **alors**
  $creerPont(N, Origine(f))$
  $emprunterArc(f, N, Arc(N, Origine(i)))$
sinon
  $emprunterArc(f, N, Arc(N, i))$
fin si

Algorithme 16 *emprunterArc*

Entrée : f, une fourmi, N, le nœud où se trouve la fourmi, A l'arc que la fourmi doit emprunter

$N \leftarrow A(N)$
si $N = Origine(f)$ **alors**
  $E(N) \leftarrow E(N) + E(f)$
  $E(f) \leftarrow 0$
  $mode(fi) \leftarrow chercheEnergie$
sinon
  si $E(N) \geq E_a$ **alors**
    $E(N) \leftarrow E(N) - E_a$
    $E(f) \leftarrow E(f) + E_a$
  sinon
    $E(f) \leftarrow E(f) + E(N)$
    $E(N) \leftarrow 0$
  fin si
fin si

Algorithme 17 *traiteArcs*

Entrée : A_c, l'ensemble des arcs au cycle c avec $|A_c| = n$
Sortie : A_{c+1}, l'ensemble des arcs au cycle c+1

$A_{c+1} \leftarrow \emptyset$
pour i **in** $1..n$ **faire**
  $\varphi_c(a_i) \leftarrow \varphi_c(a_i) \times (1 - \delta)$
  si $\varphi_c(a_i) \leq 0$ **alors**
    $A_{c+1} \leftarrow A_{c+1} \cup a_i$
  fin si
fin pour

Si ω_{f_i}, la durée de vie de la fourmi f_i est inférieure à ω, la durée de vie d'une fourmi, cette fourmi est conservée au cycle suivant. Noeud(f) est le nœud sur lequel la fourmi f se trouve.

Algorithme 18 *traiteFourmis*

Entrée : P_c, la population des fourmis au cycle c avec $|P_c| = n$
Sortie : P_{c+1}, la population des fourmis au cycle c + 1

$P_{c+1} \leftarrow \emptyset$
pour i **in** $1..n$ **faire**
  $\omega_{f_i} \leftarrow \omega_{f_i} + 1$
  si $\omega_{f_i} < \omega$ **alors**
    $P_{c+1} \leftarrow P_{c+1} \cup f_i$
  sinon
    $E(Noeud(i)) \leftarrow E(Noeud(i)) + E(i)$
  fin si
fin pour