Int. J. of Web Engineering and Technology, Vol. x, No. x, xxxx
Authors : D. Schwab, J. Goulian and A. Tchechmedjiev
Short tiltle : **Evaluation of AI methods for unsupervised WSD**

Worst-case Complexity and Empirical Evaluation of Artificial Intelligence Methods for Unsupervised Word Sense Disambiguation

Didier Schwab*
LIG (Laboratory of Informatics of Grenoble)
GETALP (Study Group for Machine Translation and Automated Processing of Languages and Speech)
Grenoble University, France
E-mail: Didier.Schwab@imag.fr
*Corresponding author

Jérôme Goulian
LIG (Laboratory of Informatics of Grenoble)
GETALP (Study Group for Machine Translation and Automated Processing of Languages and Speech)
Grenoble University, France
E-mail: Jerome.Goulian@imag.fr

Andon Tchechmedjiev
LIG (Laboratory of Informatics of Grenoble)
GETALP (Study Group for Machine Translation and Automated Processing of Languages and Speech)
Grenoble University, France
E-mail: Andon.Tchechmedjiev@imag.fr

Abstract: Word Sense Disambiguation (WSD) is a difficult problem for NLP. Algorithm that aim to solve the problem focus on the quality of the disambiguation alone and require considerable computational time. In this article we focus on the study of three unsupervised stochastic algorithms for WSD: a Genetic Algorithm (GA) and a Simulated Annealing algorithm (SA) from the state of the art and our own Ant Colony Algorithm (ACA). The comparison is made both in terms of the worst case computational complexity and of the empirical performance of the algorithms in terms of F_1 scores, execution time and evaluation of the semantic relatedness measure. We find that the worst-case complexity of GA is a factor of 100 higher that SA. However, it is difficult to make any comparison to ACA. We estimate the best parameters manually for SA and GA, but automatically for ACA (made possible by its short execution time). We find that ACA leads to a shorter execution time (factor of 10 and 100, respectively) as well as better results. Using different voting strategies, we find a small increase in the F_1 scores of SA and GA and significant improvements in the results of the ACA. With the latter, we surpass the First Sense baseline and come close to the results of supervised systems on the coarse-grained all words task from Semeval 2007.

Keywords: Unsupervised Word Sense Disambiguation, Semantic Relatedness, Ant Colony Algorithms, Stochastic optimization algorithms

Reference to this paper should be made as follows: Schwab, D., Goulian, J. and Tchechmedjiev, A. (xxxx) 'Worst-case Complexity and Empirical Evaluation of Artificial Intelligence Methods for Unsupervised Word Sense Disambiguation', *Int. J. of Web Engineering and Technology*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Didier Schwab received his PhD in Informatics in 2005 from the University of Montpellier, France. He was scientific fellow at University Sains Malaysia, Penang in 2006-2007. He has been assistant professor then associate professor at University of Grenoble 2 (University Pierre Mendes France), France since 2007. He is researcher at the Study Group for Machine Translation and Automated Processing of Languages and Speech of the Laboratory of Informatics of Grenoble. His research focuses on the representation, acquisition and exploitation of meaning including Word Sense Disambiguation.

Jérôme Goulian received his PhD in Informatics from the University of South Brittany, France. He has been an Associate Professor at University of Grenoble 2 (University Pierre Mendes France), France since 2003. He is researcher at the Study Group for Machine Translation and Automated Processing of Languages and Speech of the Laboratory of Informatics of Grenoble. His research interests include speech understanding, corpus linguistics, robust dependency parsing and more recently Word Sense Disambiguation.

Andon Tchechmedjiev is currently a first year PhD Student in the Study Group for Machine Translation and Automated Processing of Languages and Speech of the Laboratory of Informatics of Grenoble (GETALP-LIG), working on multilingual word sense disambiguation and lexical resources.

1 Introduction

Word Sense Disambiguation (WSD) is a core task in Natural Language Processing (NLP), as it has the potential to improve many of its applications: multilingual information retrieval, automatic summarization or machine translation. More specifically, the aim of WSD is to assign for each word of a text the appropriate sense(s) in context from a pre-definied sense inventory. For example, in "*The mouse is eating cheese.*", for the word ‹*mouse*›, a WSD algorithm should choose the sense that corresponds to the animal rather than the pointing device. There exist many methods to perform WSD that are commonly classified into supervised and unsupervised methods. The former are based on machine learning techniques that use a set of (manually) labelled training data, whereas the latter do not.

For the evaluation of such systems, campaigns are regularly organized in the form of the Semeval evaluation workshop that started as Senseval-1 (Kilgarriff and Kilgarriff, 1998) and has been organized regularly thereafter. Semeval has two main tasks for WSD: Lexical sample, where annotated verbs and nouns serve as a gold standard to evaluate the system; All-words, where a corpora of annotated texts (by lexicographers) serves to evaluate the system.

The main issue with supervised methods is that they require large amounts of annotated instances that are difficult to build. The sense selection for each word is made using a separate classifier, that each requires to be trained separately. Thus, most supervised systems are evaluated on the lexical sample task (Navigli, 2009), while unsupervised methods that use a pre-defined sense inventory are evaluated for the most part on the all-words task.

Our research is focussed on the application of WSD in the most general context (full text), and thus on the all-words tasks. Consequently, this article focusses on an unsupervised knowledge-based approach for WSD. More specifically, our approach is derived from the one proposed by (Lesk, 1986). This approach uses a similarity measure (between senses) that corresponds to the number of overlapping words in the definitions of the two senses. With this metric, one can select, for a given word of a text, the sense that yields the highest relatedness to a certain number of its neighbouring words (with a fixed window size). Works such as (Pedersen et al., 2005) use a Brute-Force (BF) global algorithm that evaluates the relatedness between each sense of each word and its combination to all the senses of the other words within the considered window. The execution time is exponential in the size of the input, thus reducing the maximum possible width of the window. The problem can become intractable even on the scale of short sentences: a linguistically motivated context, such as a paragraph for instance, can not be handled. Thus, such an approach can not be used for real time applications (image retrieval, machine translation, augmented reality).

In order to overcome this problem and to perform WSD faster, we are currently studying other methods than the BF approach. In this paper, we focus on three methods that globally extend a local algorithm based on semantic relatedness to the scale of a whole text. We consider two algorithms from the state of the art, a Genetic Algorithm (GA) (Gelbukh et al., 2003) and Simulated Annealing algorithm (SA) (Cowie et al., 1992); as well as an adaptation of an Ant Colony Algorithm (ACA) (Schwab et al., 2011). The aim of this article is to provide a worst-case complexity and an empirical comparison of the ant colony algorithm with the two others unsupervised methods, using the Semeval-2007, Task-7, Coarse grained corpus (Navigli et al., 2007). We will focus both on a qualitative comparison (F_1 measure) and on execution speed. Moreover, a comparison with the BF exhaustive algorithm will be presented.

After a brief review of the state-of-the-art of WSD, the algorithms are described and compared from a worst-case computational complexity stand point. Subsequently, their implementations are discussed, as well as the estimation of the best parameters. We then evaluate the algorithms on the all words task from Semeval 2007. The results are first analysed between the three algorithms and then compared to systems that participated in the Semeval task and recent systems evaluated on the same corpus and with the same scorer.

2 Brief State of the art of Word Sense Disambiguation

WSD consists in choosing the best sense among all possible senses for all words in a text. There exist many methods to perform WSD. One can read (Ide and Véronis, 1998) for works before 1998 and (Agirre and Edmonds, 2006) or (Navigli, 2009) for a complete state of the art.

2.1 Supervised WSD

Supervised WSD algorithms are based on the use of a large set of labelled training data. The training data is used to build a classifier which can determine the right sense(s) for a given word in a given context. Supervised methods such as decision lists, decision trees, naive Bayesian classification, neural networks, instance based learning (k-Nearest Neighbours, ...), Support Vector machines, *etc.* have all been applied to WSD. Even though supervised methods tend to give better results (on English) than unsupervised methods both for speed and quality, their main disadvantage is that they require a large amount of hand-annotated corpora, which are a somewhat rare and expensive resource (knowledge acquisition bottleneck) and which must be created for each sense inventory, each language and even each

specialized domain. This point, that we share with (Navigli and Lapata, 2010) led us to focus more specifically on unsupervised methods.

2.2 Unsupervised WSD

Unsupervised WSD approaches do not require the use of annotated training data. Some of these methods use non-annotated corpora to build word vectors or graphs, while others use external knowledge sources (dictionaries, thesauri, lexical databases, …). These methods aim at providing a score based on the proximity of the semantic content of two compared linguistic items (usually words or word senses). The scores can be similarity values (in the mathematical sense and therefore have a value between 0 and 1), distances or more generally any unbounded positive number.

These measures include measures or distances between vectors (LSA Deerwester et al. (1990), conceptual vectors (Schwab, 2005*a*)), measures based on taxonomic distance in a lexical network (number of edges in the graph between two senses) – some of then using all the relations in the network (Hirst and St-Onge, 1998) or a subset of the relations (Rada et al., 1989), (Leacock and Chodorow, 1998), (Wu and Palmer, 1994) –, measures based on content information (Resnik, 1995), (Lin, 1998), (Seco et al., 2004) or hybrid measures combining several of these approaches (Li et al., 2003) (Pirró and Euzenat, 2010). Readers may consult (Pedersen et al., 2005), (Cramer et al., 2010), (Tchechmedjiev, 2012) or (Navigli, 2009) for a more complete overview.

In WSD, these measures are used locally between two senses, and are then applied on a global level. In this paper we use such a measure as a local score (see section 3.1).

An intermediate category uses some annotated data as, for example, a default sense from a annotated corpus when the main algorithm fails (Navigli, 2009).

2.3 Lexical Resources

WordNet (Fellbaum, 1998) is a lexical database for English widely used in the context of WSD. WordNet is organised around the notion of ”synonym sets” (*synsets*), which represent a word, its class (noun, verb, ...) and its connections to all semantically related words (synonyms, antonyms, hyponyms,...), as well as a set of textual definition of each corresponding synset. The latest version of WordNet (3.1) contains over 155000 words for 117000 synsets.

In this article we place ourselves in a monolingual setting and use Wordnet 2.1 (required by the evaluation task).

A new resource called Babelnet is increasingly popular in the community ever since the release of the first complete version in June 2012. Babelnet (Navigli and Pozetto, 2012) is a large scale multilingual lexical resource built from the automatic mapping between Wordnet synsets and Wikipedia pages. Babelnet is based on the concept of a *Babel Synset*, that contains a Wordnet synset and set of related Wikipedia pages. The latter include Wikipedia pages mapped to wordnet synsets (through a disambiguation algorithm), pages related to the mapped pages (inter-page hyperlinks), as well as corresponding multilingual pages obtained through Wikipedia’s inter-language links and their redirections in the Wikipedias in each respective language. Babelnet is a very convenient resource in the context of multilingual WSD. Furthermore, it also contains large amount of English monolingual data and has been shown to lead to improvements for monolingual WSD as well (Navigli and Pozetto, 2012).

3 Local and global WSD algorithms

3.1 Our local algorithm : a variant of the Lesk algorithm

Our local algorithm is a variant of the Lesk algorithm (Lesk, 1986). This algorithm, proposed more than 25 years ago is very simple. It only requires a dictionary and involves no training. The score given to a sense pair is the number of words in common (considered as the text between two spaces) in the two definitions, without taking into account either the word order (bag of word approach) or syntactic or morphological information. Variants of this algorithm are still today among the best on English language corpora (Ponzetto and Navigli, 2010).

Our local algorithm exploits the links provided by WordNet: it considers not only the definition of a sense but also the definitions of the linked senses (we use all the semantic relations provided in WordNet) as done in (Banerjee and Pedersen, 2002). In the next sections, we will call this local measure $Lesk_{ext}$.

The algorithm that computes the number of overlapping words between two definitions, has a complexity of $O(n \times m)$ where n and m are the respective lengths of definitions. Given that string comparisons are relatively expensive, calculating the overlap is slow. A naive approach to mitigate the speed problem would be to pre-calculate the matrix of similarities between all word sense pairs. On the one hand, this idea is unrealistic given the potential size of dictionaries (up to several million definitions). On the other hand, our ant colony global algorithm uses "dynamic definitions" that change at execution time (see section 5), thus rendering the pre-calculated values obsolete.

In this work we optimise the calculation by using a pretreatment that takes place in two stages. First, an integer is assigned to each word found in the dictionary. Then, each definition is converted in a vector of numbers corresponding to the words it contains, sorted from smallest to largest. We call these vectors, definition vectors.

This conversion has two advantages: comparing numbers is much more efficient than comparing strings. Sorting these values avoids unnecessary comparisons and lets us gain in efficiency. Thus, with this pretreatment, the complexity decreases from $O(n \times m)$ to $O(n)$ where $n \geq m$.

3.2 Global algorithms

A global algorithm is a method that allows to extend a local algorithm to a whole text in order to infer a sense for each word. The most direct algorithm is the exhaustive (BF) method, used for example in (Banerjee and Pedersen, 2002). It considers the combinations of all senses of the words in a given context (word window or text) in order to assign a score to each combination and then to choose the combination with the best score. The main problem of this method is the rapid combinatorial explosion it creates. For example, in "*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.*" which is made of 17 words, ‹*picture*› and ‹*air*› have 9 senses, ‹*paint*› 4, ‹*be*›, ‹*point*›, ‹*figure*› 13, ‹*flat*› 17, ‹*very*›, ‹*often*› 2, ‹*ground*›, ‹*foot*› 11, ‹*look*› 10, ‹*stand*› 12, ‹*at all*›, ‹*downwards*› 1 , ‹*hang*› 15. There are 137,051,946,345,600 combinations to explore, which is almost the number of operations 3300 Intel Core i7-990X processors (3,46GHz, 6 cores, 12 threads) can perform in one second. Hence, the BF method is very difficult to apply in real conditions and moreover, makes the use of a longer analysis context impossible.

To circumvent this problem, several approaches are possible. The first, called *complete approaches*, try to reduce combinatorics by using pruning techniques and heuristics of choice. In the context of WSD, this is the case, for example, of the approach proposed by (Hirst and St-Onge, 1998), based on lexical chains (a taxonomic semantic similarity measure

based on the overall relations of WordNet), that combine restrictions[1] during the construction of the global lexical chain with a greedy heuristic. According to (Navigli, 2009) the major problem of this approach is its lack of precision caused by the greedy strategy used. However, various improvements have been proposed among which those proposed by (Silber and McCoy, 2000). Other interesting complete approaches were conducted in the context of unsupervised word sense disambiguation, in particular (Brody and Lapata, 2008). The second ones, called *incomplete approaches* as they explore only a part of the search-space, use heuristics to guide them to areas of the search-space that are more promising. These heuristics are generally based on probabilities: choices are made stochastically, which is what interests us here.

We can then distinguish two main methods:

- neighborhood approaches (new configurations are created from existing configurations) among which are approaches from artificial intelligence such as genetic algorithms or optimization methods such as simulated annealing ;

- constructive approaches (new configurations are generated by adding iteratively new elements of solutions to the configurations under construction), among which are for example ant algorithms.

3.3 Context of this work

The aim of this paper is to compare our ant colony algorithm (incomplete and constructive approach) to others incomplete approaches. We choose to first confront our approach to two neighborhood approaches that have been used in the context of unsupervised word sense disambiguation: genetic algorithms (Gelbukh et al., 2003) and simulated annealing (Cowie et al., 1992).

In this paper, we choose to consider a whole text as the context window to be disambiguated. This choice is also made by (Cowie et al., 1992) for their simulated annealing algorithm and by (Gelbukh et al., 2003) for their genetic algorithm; the idea being taken up recently by (Navigli and Lapata, 2010). Many approaches, however, use a smaller context.

From our point of view, this leads to two problems. The first is that we have no way to ensure consistency between the senses selected. Two generally incompatible senses can be chosen by the algorithm because the context does not include the second word that can make the difference. For example, even with a window of six words before and six words after, the sentence "*The two planes were parallel to each other. The pilot had parked them meticulously.*", "pilot" does not help to disambiguate the term "planes". The second problem is that text usually keeps some semantic unity. For example, as noted by (Gale et al., 1992) or (Hirst and St-Onge, 1998), a word used several times in a text has generally the same sense ; this information can not be exploited within a word window.

This is the reason why (presence of a reduced-size word window) some approaches such as the application to WSD done by (Mihalcea et al., 2004) of the *PageRank* (Brin and Page, 1998), that although uses graph search-space as the ant colony algorithm, have been excluded from this study.

Moreover, the aim of this article is not to compare the quality of these three incomplete approaches to the optimal solution. Indeed, one of our perspectives is to work on applications that require real-time disambiguation. Now, our first experiments (Schwab et al., 2011) show that the exhaustive algorithm, using our local semantic measure but evaluated for lack of anything better (combinatorial reasons) on a reduced context (the sentence), was be able to provide an optimum only on 77% of the text. The reader can find in (Schwab et al., 2011) all the results that can asses the quality of the disambiguation of our ant colony algorithm compared to the exhaustive algorithm.

We therefore focus on the execution time of each of these stochastic approaches taking a whole text as our context of analysis. Moreover, in order to compare our algorithm to genetic algorithms and simulated annealing, we place ourselves, as far our ant algorithm is concerned, in a simple environment (basic hierarchal structure of the text).

4 Global stochastic algorithms for WSD

Two of the algorithms we study are both stochastic optimisation methods and share common mechanics and instance representations. We first present the representation of the WSD problem as configurations for simulated annealing and genetic algorithms followed by the calculation of a fitness value for these configurations. Then, we describe the two algorithms in general terms.

4.1 Instance representation for WSD

Both GA and SA work on the same representation for the configurations (or **instances**) in the search space. We use vectors where each index represents each of the features (dimensions) of the search space. The value associated to each index of this feature vector is discrete.

The aim of WSD is to associate to each word w_i of a text of *m* words one of its possible senses $w_{i,j}$ from a sense inventory (Princeton WordNet 2.1 here). Each word of the text constitutes a feature of the search space and thus for a text of length *m*, the size of the instance is *m* as well. Each index of the feature vector takes its values among the possible senses associated to the corresponding word according to the sense inventory. For each word, the possible senses are represented by their order of appearance in the sense inventory, such that the first sense is represented by one, the second by two and so forth.

Thus each feature vector or configuration or instance corresponds to a unique combination of word senses for the whole text. In our specific implementation, a configuration C is an array of *m* integers such that $j = C[i]$ is the selected sense j of a word of the text w_i at the i^{th} position.

4.2 Global score or fitness value of the configurations

Both algorithms require some *fitness* measure to evaluate how good a configuration is. With this in mind, the score of the selected sense of a word can be expressed as the sum of the local scores between that sense and the selected senses of all the other selected senses for the words of the text. In order to obtain a *fitness* value (*global score*) for the whole configuration, (Cowie et al., 1992) and (Gelbukh et al., 2003) both use a sum of the scores for all selected senses of the words of the text: $Score(C) = \sum_{i=1}^{m} \sum_{j=i}^{m} Lesk_{ext}(w_{i,C[i]}, w_{j,C[j]})$. The complexity of this algorithm is hence $O(m^2)$ where m is the number of words in the text.

4.3 Genetic algorithm

The genetic algorithm based on (Gelbukh et al., 2003) can be divided into five phases: initialisation, selection, crossover, mutation and evaluation. During each generation, the algorithm goes through each phase but initialisation, in the same order as above.

The initialisation phase consists on the generation of a random initial population of λ individuals (λ configurations of the problem). After each generation, the new population size is constant to λ.

During the selection phase, the score of each individual of the current population is computed. A crossover ratio (*CR*) is used to determine which individuals of the current population will be selected for crossover. The probability of an individual being selected is *CR* weighted by the ratio of the score of the current individual over that of the best individual. Individuals who are not selected for crossover are merely cloned (copied) into the new generation. Additionally the best individual is systematically added to the next generation.

The crossover phase sorts the individuals on their global score. If the number of individuals is odd, the individual with the lowest score is unselected and cloned into the new population as it cannot serve for crossover. The crossover operator is then applied to the individuals two by two in decreasing order of their score. That is, the two configurations are swapped around two random pivots (everything but what is between the pivots is swapped).

During the mutation phase, each individual has a probability of mutating (parameter *MR*, for Mutation Rate). A mutation corresponds to *MN* random changes in the configuration.

The evaluation phase corresponds to the evaluation of the termination criteria, which is here the convergence of the score of the best individual. In order to check for the convergence, a stagnation threshold *STH* is used. In other words if the score of the best individual remains the same for *STH* generations, the algorithm terminates.

4.4 Simulated annealing

The simulated annealing approach as described in (Cowie et al., 1992) is based on the physical phenomenon of metal cooling.

Simulated annealing works with the same configuration representation as the genetic algorithm, however it uses a single configuration randomly initialised. The algorithm is organised in cycles and in iterations, each cycle being composed of *IN* iterations. The other parameters are the initial temperature T_0 and the cooling rate $ClR \in [0;1]$. The choice of the initial temperature passes through the choice of the initial acceptance probability. However, in the experiments described section **Error! Reference source not found.**, we place ourselves exactly in the same context than (Cowie et al., 1992) (same decreased probabilities table) and have taken their initial hypothesis. At each iteration a random change is made to the current configuration C_c, which results in a new configuration $C_c^{'}$. Given that $\Delta E = Score(C_c) - Score(C_c^{'})$, the probability $P(A)$ of acceptance of configuration $C_c^{'}$ replaces C_c is:

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

The reason why configurations with lower scores have a chance to be accepted, is to prevent the algorithm from converging on a local maximum. Lower score configurations allow to explore other parts of the search-space which may contain the global maximum.

At the end of each cycle, if the current configuration is the same as the configuration at the end of the previous cycle, then the algorithm terminates. Otherwise, the temperature is lowered to $T \cdot ClR$. In other words, the more cycles it takes for the algorithm to converge, the lower the probability to accept lower score configurations: this guarantees the algorithm converges at some point. The configuration with the maximal score is saved and taken as a result regardless of the convergence configuration.

5 Global Ant Colony Algorithm

5.1 Ant Colony Algorithm

Ant colony algorithms come from biology and from observations of ant social behavior. Indeed, these insects have the ability to collectively find the shortest path between their nest and a source of food (or energy). It has been demonstrated that cooperation inside an ant colony is self-organised and emerges from interactions between individuals. These interactions are often very rudimentary and allow the colony to solve complex problems. This phenomenon is called swarm intelligence (Bonabeau and Théraulaz, 2000) and is increasingly popular in computer science where centralised control systems are often

successfully replaced by other types of control based on interactions between simple elements.

Artificial ants have first been used for solving the Traveling Salesman Problem (Dorigo and Gambardella, 1997). In these algorithms, the environment is usually represented by a graph, in which virtual ants exploit pheromone trails deposited by others, or pseudo-randomly explore the graph.

These algorithms are a good alternative for the resolution of problems modeled with graphs. They allow a fast and efficient exploration close to other search methods. Their main advantage is their high adaptivity to changing environments. Readers can refer to (Dorigo and Stützle, 2004), (Monmarche et al., 2009) or (Guinand and Lafourcade, 2010) for a state of the art.

5.2 Ant colony Algorithm for Word Sense Disambiguation

5.2.1 Principle

The environment of the ant colony algorithm is a graph that can be linguistic: a morphological (Rouquet et al., 2010) or morpho-syntactic lattice (Schwab and Lafourcade, 2007), or simply organised following the structure of the text (Guinand and Lafourcade, 2010).

Depending on the environment chosen, the results of the algorithm differ. We are currently investigating this aspect, but as the focus of our article is to make a comparison between ACA and the two other methods presented earlier, we will use a simple graph following the structure of the text (see Fig. 1) that uses no external linguistic information (no morpho-syntactic links within a sentence for example).

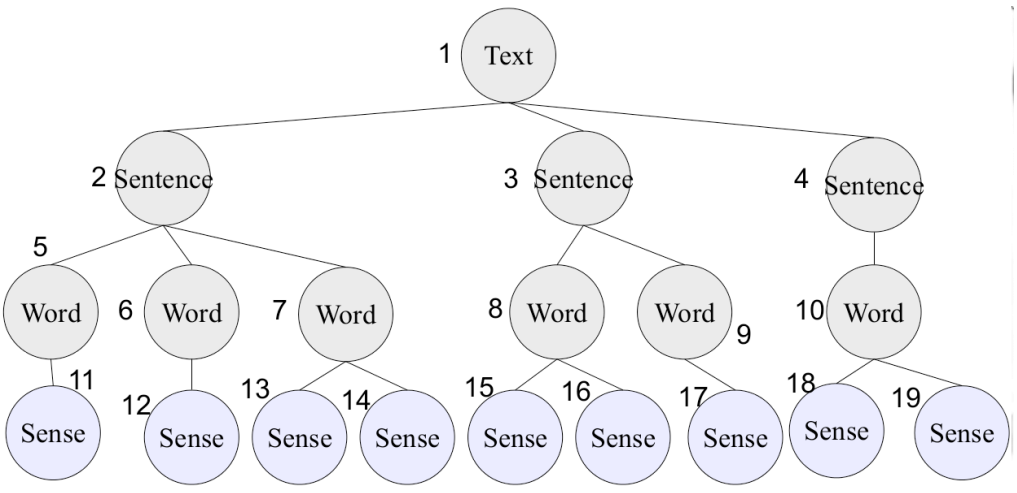


Figure 1: The environment for our experiment: text, sentences and words correspond to common nodes (1-10) and word senses to nests (11-19)

In this graph, we distinguish two types of nodes: nests and plain nodes. Following (Schwab, 2005*b*) or (Guinand and Lafourcade, 2010), each possible word sense is associated to a nest. Nests produce ants that move in the graph in order to find energy and to bring it back to their mother nest: the more energy is deposited by ants, the more ants can be produced by the nest in turn. Ants carry an odour (array) that contains the words of the

definition of the sense of its mother nest. From the point of view of an ant, a node can be: (1) *its mother nest*, where it was born; (2) *an enemy nest* that corresponds to another sense of the same word; (3) *a potential friend nest*: any other nest; (4) a *plain node*: any node that is not a nest. Furthermore, to each plain node is also associated an odour vector of a fixed length that is initially empty. For example, in Fig. 1, for an ant born in nest 19: nest 18 is an enemy (as their are linked to the same word node, 10), its potential friend nodes are from 11 to 17 and common nodes are from 1 to 10.

Ant movements are function of the scores given by the local algorithm (cf. Section 3.1), of the presence of energy, of the passage of other ants (when passing on an edge ants leave a pheromone trail that evaporates over time) and of the odour vectors of the nodes (ants deposit a part of their odour on the nodes they go through). When an ant arrives on a potential friend nest (that corresponds to a sense of another terms), it can either continue its exploration or, depending on the score between this nest and its mother nest, decide to build a bridge between them and to follow it home. Bridges behave like normal edges except that if at any given time the concentration of pheromone reaches 0, the bridge collapses.

Depending on the lexical information present and the structure of the graph, ants will favour following bridges between more closely related senses. Thus, the more closely related the senses of the nests are, the more a bridge between them will contribute to their mutual reinforcement and to the sharing of resources between them (thus forming *meta-nests*); while the bridges between more distant senses will tend to fade away. We are thus able to build (constructive approach) interpretative paths (possible interpretation of the text) through emergent behaviour and to suppress the need to use a complete graph that includes all the links between the senses from the start (as is usually the case with classical graph-based optimisation approaches).

5.2.2 Implementation details

In this section we first present the notations used (Table 1) as well as the parameters of the Ant Colony Algorithm and their typical value ranges (Table 2), followed by a detailed description of the various steps of the algorithm.

Notation	Description
F_A	Nest that corresponds to sense A
f_A	Ant born in nest F_A
$V(X)$	Odour vector associated with X (ant or node)
$E(X)$	Energy on/carried by X (ant or node)
$Eval_f(N)$	Evaluation of a node N by an ant f
$Eval_f(A)$	Evaluation of an edge A (quantity of pheromone) by an ant f
$\varphi_{(t)}(A)$	Quantity of pheromone on edge A at given moment/cycle t

Table 1: Main notations for the Ant Colony Algorithm

Notation	Description	Value
E_a	Energy taken by an ant when it arrives on a node	1-30
E_{max}	Maximum quantity of energy an ant can carry	1-60
δ_ϕ	Evaporation rate of the pheromone between two cycles	0.0-1.0
E_0	Initial quantity of energy on each node	5-60
ω	Ant life-span	1-30 (cycles)
L_V	Odour vector length	20-200
δ_V	Percentage of the odour vector components (words) deposited by an ant when it arrives on a node	0-100%
c_{ac}	Number of cycles of the simulation	1-500

Table 2: Parameters of the Ant Colony Algorithm and their typical value-ranges

5.2.3 Simulation

The execution of the algorithm is a potentially infinite succession of cycles. After each cycle, the state of the environment can be observed and used to generate a solution. A cycle is composed of the following actions: (1) eliminate dead ants and bridges with no pheromone; (2) for each nest, potentially produce an ant; (3) for each ant: determine its mode (energy search or return); make it move; potentially create an interpretative bridge; (4) update the environment (energy levels of nodes, pheromone and odour vectors).

Ant production, death and energy model

Initially, we assign a fixed quantity of energy E_0 to each node of the environment. At the beginning of each cycle, each nest node F_A has an opportunity to produce an ant f_A using 1 unit of energy, with a probability $P(F_A)$. In accordance with (Schwab and Lafourcade, 2007) or (Guinand and Lafourcade, 2010), we define it as the following sigmoid function (inspired by nature and often used with artificial neural networks (Lafourcade, 2011)): $P(F_A) = \frac{arctan(E(F_A))}{\pi} + 0.5$ (Figure 2).

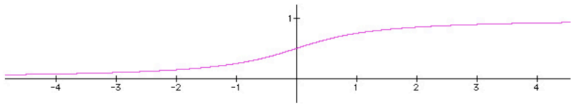


Figure 2: Plot of $arctan(x)/\pi + 0.5$

When created, an ant has a lifespan of ω cycles (see Table 2). When the life of an ant reaches zero, the ant is deleted at the beginning of the next cycle and the energy it carried is deposited on the node where it died. By thus doing, we ensure that the global quantity of energy in system remains the same; this plays a fundamental role in the convergence (monopolization of resources by certain nests) to a solution.

Ant movements

The ants' movements are random, but influenced by the environment. When an ant is on a node, it assigns a transition probability to the edges leading to all neighbouring nodes. The

probability to cross through an edge A_j in order to reach a node N_i is $P(N_i, A_j) = \frac{Eval_f(N_i, A_j)}{\sum_{k=1,l=1}^{k=n,l=m} Eval_f(N_k, A_l)}$ where $Eval_f(N, A) = Eval_f(N) + Eval_f(A)$ is the evaluation function of a node N when coming from an edge A.

A newborn ant seeks food. It is attracted by the nodes which carry the most energy ($Eval_f(N) = \frac{E(N)}{\sum_0^m E(N_i)}$), but avoids to go through edges with a lot of pheromone, $Eval_f(A) = 1 - \varphi_t(A)$ in order to favour a greater exploration of the search space. The ant collects as much energy as possible until it decides to bring it back home (return mode) with the probability $P(return) = \frac{E(f)}{E_{\max}}$. Consequently, when the ant reaches its carrying capacity, the probability to switch to return mode is 1. Then, it moves while following (statistically) the edges that contain the most pheromone, $Eval_f(A) = \varphi_t(A)$ and leading to nodes with an odour vector close to their own, $Eval_f(N) = \frac{Lesk_{ext}(V(N), V(f_A))}{\sum_{i=1}^{i=k} Lesk_{ext}(V(N_i), V(f_A))}$.

Creation, deletion and types of bridges

When an ant arrives on a node adjacent to a potential friend nest (i.e. that corresponds to a sense of a word), it has to decide between taking any of the possible paths or to go on that nest node. As such, we are dealing with a particular case of the ant path selection algorithm presented above in Section , with $Eval_f(A) = 0$ (the pheromone on the edge is ignored). The only difference is that if the ant chooses to go on the potential friend nest, a bridge between that nest and the ant's home nest can be built. The ant can then follow the bridge to go home. Bridges behave like a regular edges, except that if the concentration of pheromone on them reaches 0, they collapse and are removed.

Pheromone Model

Ants have two types of behaviours: they are either looking to gather energy or to return to their mother nest. When they move in the graph, they leave pheromone trails on the edges they pass through. The pheromone density on an edge influences the movements of the ants: they prefer to avoid edges with a lot of pheromone when they are seeking energy and to follow them when they want to bring the energy back to their mother nest.

When passing on an edge A, ants leave a trail by depositing a quantity of pheromone $\theta \in \mathbb{R}^+$ such that $\varphi_{t+1}(A) = \varphi_t(A) + \theta$.

Furthermore, at the end of each cycle, there is a slight (linear) evaporation of pheromones (penalizing little frequented paths). Thus, $\varphi_{t+1}(A) = \varphi_t(A) \times (1 - \delta_\phi)$ where δ_ϕ is the pheromone evaporation rate.

Odour

The odour of a nest is the numerical sense vector (as introduced in Section 3.1) and corresponds to the definition of the sense associated to the nest. All ants born in the same nest have the same odour vector. When an ant arrives on a common node N, it deposits some of the components of its odour vector (following a uniform distribution), which will be added to or will replace existing component of the node's vector $V(N)$. However, the odour of nest nodes is never modified.

This mechanism allows ants to find their way back to their mother nest. Indeed, the closer a node is to a given nest, the more ants from that nest will have passed through and deposited odour components. Thus, the odour of that node will reflect its nest neighbourhood and allow ants to find their way by computing the score between their odour (that of their mother nest) and the surrounding nodes and by choosing to go on the node yielding the highest score. This process leaves some room for error (such as an ant arriving on a nest other than its own), which is beneficial as it leads ants to build more bridges (see Section).

5.3 An illustrated example

Through the journey of a single ant (Figures 3, 4, 5), we illustrate the different steps of the algorithm on the simple sentence "mice pilot computers". The definition vectors are represented under their corresponding node and are preceded by the letter V. The energy levels are represented as vertically stacked bars preceded by the letter E. The lines between the nodes represent the paths and their thickness the pheromone concentration. On Figures 4 and 5 the circled numbers show the succession of steps of the ant. At each step the ant will be represented by the image of an ant. The quantity of energy the ant carries is represented by the number of bars under to the $E(A)$ label. The numbers in the vectors under the nest nodes correspond to the indexed and sorted words of the definitions of their respective senses.

First, in Figure 3 we can see an ant being created in the node `mouse#n#1`.

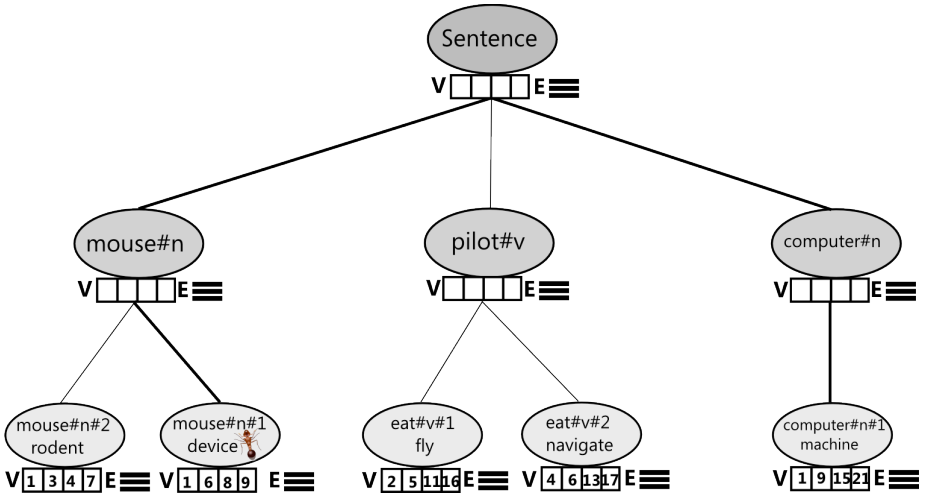


Figure 3: Running Example: starting state of the environment

Then, on Figure 4, the ant starts in exploration mode (1), takes an unit of energy from its nest and begins to explore the graph by first going on the `mouse#n` word node (2), its only choice. Upon arriving on `mouse#n` it takes one bar of energy and deposits a pheromone on the path between its nest and the parent word node. Furthermore, it randomly chooses to deposit two components of the definition of its home nest in the vector of `mouse#n`. Next, the ant chooses to follow path (3), where it deposits a unit of pheromone when it passes through to reach the sentence root node (4). There, the ant deposits again two random components of its nest's definition vector, and takes a bar of energy.

Now, the ant carries three bars of energy, and through a pseudo-random decision switches to return mode. The ant now has the choice between going on `pilot#v` or on `computer#n`. Given that both vector are empty, the ant uniformly decides to follow path (5), where it deposits pheromone, before arriving at `computer#n` (6). The ant deposits components of its odour, but since the ant is in return mode, it does not take any energy.

Next, the ant has little choice, but to follow the only path available (7) that goes to `computer#n#1` (8) and deposits pheromone on the way through.

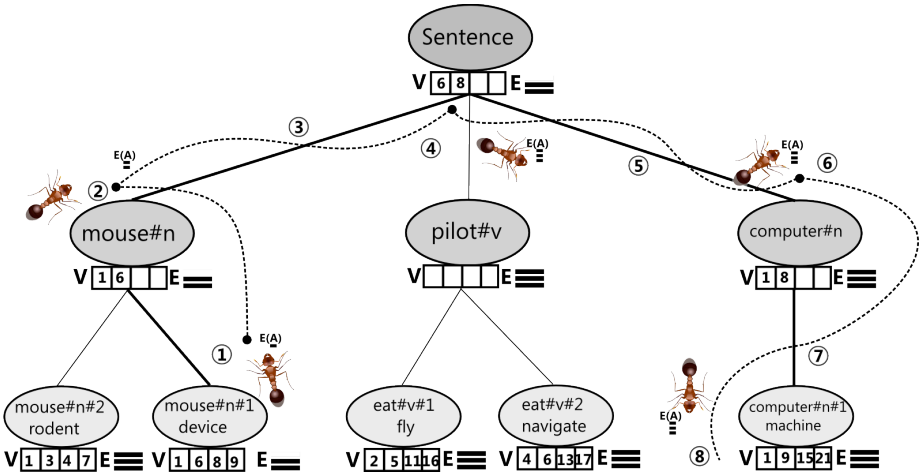


Figure 4: Running Example: Ant movement, search and return modes

Now that the ant has arrived potential friend nest, it has the possibility to build a bridge to its home nest. The choice to do so is made pseudo-randomly in Figure 5, due to the similarity between the odour vectors of `mouse#n#1` and `computer#n#1`. Next, the ant follows the bridge, where it also deposits some pheromone (9). Finally, when the ant reaches its home nest, it deposits all the energy it carries with it (10). The ant switches back to exploration mode and continues its search until it dies.

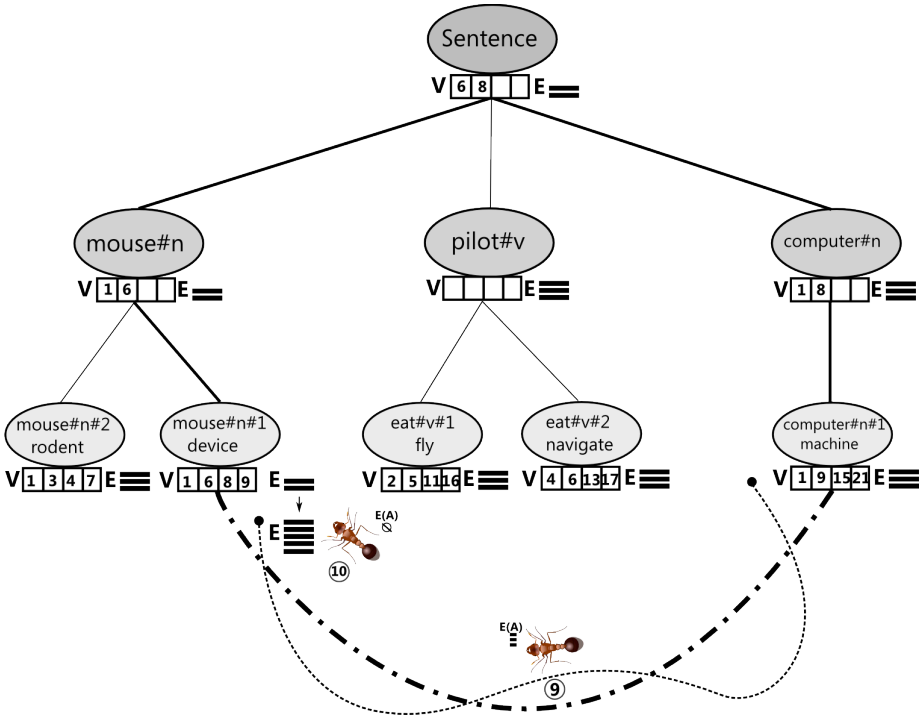


Figure 5: Running Example: Return mode and bridges

The pheromone trails left by the ants will incite other to follow them more than paths with less energy. Furthermore, from now on, any ants born on `mouse#n#1` and `computer#n#1` will have the possibility to go directly from one sense to the other through the bridge. Thus, the link can very likely be reinforced by other ants, all the more so if the bridged senses are closer.

5.4 Global Evaluation

In (Schwab and Guillaume, 2011) the final configuration was selected to be the set of nests (word sense) with the highest energy values for each word. However, here we take a different approach. At the end of each cycle, we build the current problem configuration from the graph: for each word, we take the sense corresponding to the nest with the highest quantity of energy. Subsequently, we compute the global score of the configuration (see Section 4.2). Over the execution of the algorithm we keep the configuration with the highest absolute score, which will be used at the end to generate the solution.

5.5 Main parameters

Here we present a short characterisation of the influence of the parameters on the emergent phenomena in the system:

- The maximum amount of energy an ant can carry, E_{max}, influences how much an ant explores the environment. Ants cannot go back through an edge they just crossed and have to make circuits to come back to their nest (if the ant does not die before that). The size of the circuits depend on the moment the ants switch to return mode, hence on E_{max}.

- The evaporation rate of the pheromone between cycles (δ_ϕ) is one of the memories of the system. The higher the rate is, the least the trails from previous ants are given importance and the faster interpretative paths have to be confirmed (passed on) by new ants in order not to be forgotten by the system.

- The initial amount of energy per node (E_0) and the ant life-span (ω) influence the number of ants that can be produced and therefore the probability of reinforcing less likely paths.

- The odour vector length (L_v) and the proportion of odour components deposited by an ant on a plain node (δ_V) are two dependent parameters that influence the global system memory. The higher the length of the vector, the longer the memory of the passage of an ant is kept. On the other hand, the proportion of odour components deposited has the inverse effect.

Given the lack of an analytical way of estimating the best parameters of both the Ant Colony Algorithm and the other algorithms presented, they have to be estimated experimentally, which is detailed in Section 7.

6 Theoretical Worst-case Time Complexity

As the algorithms presented here are stochastic in nature (with the exception of BF), a full fledged worst case analysis would require to determine the theoretical number of iterations necessary until convergence, and not merely calculate the worst case cost relative to the number of iterations. Since the primary concern here is the quality of the solutions obtained rather than the performance, a comparison based on a rough approximation of the worst case time complexity is enough. More specifically, a computational parallel between simulated annealing and the genetic algorithm allows for a comparison that gives an upper bound on the

ratio between their worst-case time complexities. This worst case analysis aims moreover at identifying, roughly in the context of this first study, the parameters of the algorithms that have a main influence on time execution.

It is assumed that the cost of the calculation of the local scores is $O(1)$ as the computation is the same for all algorithms and we are only interested in the global algorithms here.

In the following, we consider m as the number of words to disambiguate in the text and s as the number of possible senses in the text. We will also consider that s and m are linearly related by a constant k which is hence the average number of senses for the words of the text. For example, in the test corpus we will use in section 7.1, $k = 6.19$. At last, we denote $T(X)$, the execution time of X, which is either an algorithm or a phase of an algorithm.

6.1 Brute-Force Algorithm

The brute-force algorithm explores all possible configurations, computes their scores then takes the best one.

The number of possible configurations is the average number of senses to the power of the number of words. Assuming the average number of senses is a constant k, the number of possible configurations is k^m.

Computing the overall score is of complexity $O(m^2)$, hence the worst-case complexity of the algorithm is $O(k^m.m^2) = O(k^m)$.

6.2 Genetic Algorithm

The algorithm is composed of five phases: the initialisation phase (first generation of the population) which is done only once followed by a repetitive application of the four other phases: selection, crossover, mutation and evaluation.

- *The initialisation phase I* is merely the random generation of the configurations of the initial population (λ individuals), and the calculation of their scores. Assuming the generation of a random number costs $O(1)$, the generation of a random configuration is $O(m)$ and the cost of computing its global score is $O(m^2)$, then $T(I) = \lambda \cdot (O(m) + O(m^2)) = O(\lambda \cdot m^2)$.

- *The selection phase S* consists of computing and sorting the scores of the individuals of the population, then selecting the individuals for crossover. Individuals not selected for crossover are merely inserted into the new population unchanged. Assuming computing the score costs $O(m^2)$, the score calculation over the λ individuals costs $O(\lambda \cdot m^2)$. The sorting algorithm used is merge sort whose worst-case time complexity is $T(Sort) = O(\lambda \cdot log_2(\lambda))$. All other steps are $O(1)$ thus the worst-case time complexity of the selection phase is $T(S) = O(\lambda \cdot m^2 + \lambda \cdot log_2(\lambda)) = O(\lambda \cdot m^2)$.

- *The crossover phase C* consists of trimming selected individuals so that there is an even amount of them, after which the crossover operator is applied to the selected individuals in pairs. Looking for all the individuals costs $O(\lambda)$ and the application of the crossover operator is $O(m)$ (everything is changed). Hence, the worst-case time complexity of the crossover algorithm is $T(C) = O(\lambda \cdot m)$. The trimming of the selected individuals costs $O(1)$, and the initialisation of the offsprings costs $O(m)$. As for the loop, the selection of the crossover points is $O(1)$ and the crossover operator $O(m)$. Adding the offsprings to the new generation also costs $O(1)$. The cost of the operation before the loop is thus $O(m)$. The number of iterations being at worst λ, the overall cost of the loop is $O(\lambda \cdot m)$. Hence the worst-case time complexity of the crossover algorithm is $T(C) = O(m + \lambda \cdot m) = O(\lambda \cdot m)$.

- *In the mutation phase* M, each individual of the new generation has a probability *MR* of undergoing *MN* random changes. In the worst-case, *MR = 1* (all the individuals mutate) and *MN = m* (changes are done on each component of the configuration). The number of individuals undergoing mutations being λ, the worst-case time complexity of the mutation phase is $T(M) = O(\lambda \cdot m)$.

- *In the evaluation phase* E, we consider that convergence is reached when the score of the best individual reaches stagnation during *STH* generations. The termination check is only a single comparison that costs $O(1)$. Additionally, the new population P_{n+1} becomes the current population P_n (P_{n+1} becomes empty). The swapping is $O(1)$, and updating P_{n+1} costs in the worst case $O(\lambda)$. Thus, the evaluation phase costs $T(E) = O(\lambda)$.

Assuming the total number of generations before convergence is denoted G_n, the overall complexity of the genetic algorithm is . $O(G_n \cdot (T(I) + T(S) + T(C) + T(M) + T(E)) = O(G_n \cdot (\lambda \cdot m^2 + \lambda \cdot m^2 + \lambda \cdot m + \lambda \cdot m + \lambda)) = O(G_n \cdot \lambda \cdot m^2) = O(m^2)$.

As stated in Oliveto et al. (2007), theoretical convergence analyses only exist for simple forms of evolutionary algorithms for toy problems. Most performance evaluations on such algorithms are done in an empirical manner.

6.3 Simulated Annealing

The simulated annealing algorithm is composed of three phases: Initialisation, Evaluation and Annealing.

- *The initialisation phase* I is merely the random generation of the initial configuration, and the calculation of its score. Assuming the generation of a random number costs $O(1)$ and the calculation of the configuration score costs $O(m^2)$, then $T(I) = O(m) + O(m^2) = O(m^2)$.

- *The evaluation phase* E consists of checking whether there have been changes between the current and previous configuration (that is $\forall i \in [1; N], C_{current}(i) = C_{previous}(i)$), which costs $O(m)$, and then decreasing the temperature if the configurations are not identical (otherwise the algorithm terminates), which is $O(1)$.Thus, $T(E) = O(m)$.

- *The annealing phase* A consists of iteratively (*IN* times) applying a random change in current configuration $C_{current}$ to obtain $C'_{current}$, and then determining whether to keep the new configuration as $C_{current}$. The new configuration is kept with a probability function of the scores of $C_{current}$ and $C'_{current}$. A copy of the configuration with the highest score is saved at each step. This copy costs $O(m)$ while computing the scores costs $O(m^2)$. All other operations cost $O(1)$ and the process is repeated IN times. The worst case time complexity of the annealing phase is hence $T(A) = O(IN \cdot m^2)$.

Assuming the total number of cycles is c_{sa}, the number of times the annealing process is repeated is IN and the size of the text is m, the overall complexity of simulated annealing is . $O(T(I) + c_{sa} \cdot (T(E) + T(A)) = O(m^2 + c_{sa} \cdot (m + IN.m^2)) = O(c_{sa} \cdot IN \cdot m^2) = O(m^2)$.

As illustrated by Laarhoven and Aarts (1987, Chapter 6), there exists some general theoretical results about the convergence of the algorithm in the worst-case for simple

problems, none however for any of its applications (including WSD), nor for the average-case.

6.4 Ant colony algorithm

The algorithm is composed of iterations of four phases: initialisation, movement, evaluation and environment update.

- *The initialisation I* is merely the production, or not, of one ant by each nest. Assuming production and random number generation cost $O(1)$, $T(I) = O(s)$.
- *In the movement phase M*, each ant moves randomly or dies. Building bridges, pheromone depositing and moving cost $O(1)$, while the evaluation of the destination is function of the number of neighbouring nodes to the ant's current location. In the worst case, an ant is on a nest linked to all other nests (i.e. $s-1$ edges) and linked to its mother node (i.e. the word), therefore it is linked to s nodes. In the worst case moreover, s ants are produced at each step and their life-span is ω. Once we reach more than ω steps, there is a maximum of $\omega \cdot s$ ants. Hence, $T(M) = O((\omega \cdot s) \cdot s) = O(\omega \cdot s^2)$.
- *The evaluation phase E* consists simply in computing the score of the current configuration, hence $T(E) = O(m^2)$.
- *The update phase U* is merely the decreasing of the pheromone concentration on each edge. In the worst case, the number of edges is the highest when all nests are linked together. Updates of these edges is therefore in $O(s^2)$. The number of edges between common nodes is linearly related to the number of senses. Thus, we can consider that the update phase for these edges costs $O(s)$. Hence, the update phase costs $T(U) = O(s^2)$.

Assuming the total number of cycles is c_{ac}, the number of possible senses is s and the size of the text to disambiguate is m, the life-span of an ant is ω, the overall complexity of the ant algorithm is $O(c_{ac} \cdot (T(I) + T(M) + T(E) + T(U))) = O(c_{ac} \cdot (s + \omega \cdot s^2 + m^2 + s^2)) = \mathcal{O}(c_{ac} \cdot (\omega \cdot s^2 + m^2 + s^2))$. As the number of senses and the number of words are linearly related ($s = km$), we consider that the complexity is $O(c_{ac} \cdot (\omega \cdot k^2 m^2 + m^2 + k^2 m^2)) = O(c_{ac} \cdot \omega \cdot k^2 \cdot m^2) = O(m^2)$.

6.5 Comparison

As we can see, the brute-force algorithm has an exponential complexity while stochastic algorithms have a similar worst complexity in $O(m^2)$. They differ on the number of times the global score is computed, which in turn depends on the values of the parameters used by each algorithm. This worst case study gave us an idea of the hidden constants on which depend the execution time: $G_n \cdot \lambda$ for the genetic algorithm, $c_{sa} \cdot IN$ for simulated anneling and $c_{ac} \cdot \omega \cdot k^2$ for the ant colony algorithm. In section 7.4, we will present an experimental measure of the number of evaluations of the score function in order to see if the order of magnitude between the algorithms is preserved.

7 Empirical Evaluation

In this section we first describe the evaluation task we used to evaluate the three systems, followed by the methodology we used for the estimation of the parameters, and then the experimental protocol for the empirical quantitative comparison of the algorithms and

subsequently, the interpretation of the results. Finally, we briefly compare the number of evaluations of the semantic similarity score function ($Lesk_{ext}$) and discuss the positioning of our system relatively to the other systems that are evaluated with Semeval 2007 Task 7 (participating systems and more recent advances).

7.1 Evaluation campaign task

We evaluated the algorithms with the Semeval 2007 coarse-grained English all-words task (Navigli et al., 2007). Composed of 5 texts from various domains (journalism, book review, travel, computer science, biography), the task consists in annotating 2269 words with one of their possible senses from WordNet, with an average degree of polysemy of 6.19. The evaluation of the output of the algorithm is done considering coarse grained senses distinction i.e. close senses are counted as equivalent (e.g. snow/precipitation and snow/cover).

A Perl script that evaluates the quality of the solutions is provided with the task files and allows us to compute the **P**recision, **R**ecall, and F_1 scores, the standard measures for evaluating WSD algorithms (Navigli, 2009) (when 100% of the corpus is annotated, $P = R = F_1$), against a reference sense assignment annotated by lexicographers.

7.2 Parameter estimation

The algorithms we are interested in have a certain number of parameters that need tuning in order to obtain the best possible score on the evaluation corpus. There are three approaches:

- Make an educated guess about the value ranges based on *a priori* knowledge about the dynamics of the algorithm;
- Test manually (or semi-manually) several combinations of parameters and determine the influence of making small adjustments to the values ;
- Use a learning algorithm on a subset of the evaluation corpus, for example with SA or GA to find the parameters that lead to the best score.

Both GA and SA, as presented in (Gelbukh et al., 2003) and (Cowie et al., 1992) respectively, use the Lesk semantic similarity measure as a score metric. We reimplemented them with the $Lesk_{ext}$ measure and used the *optimal* parameters provided in the original papers. However, the similarity values are higher than with the standard Lesk algorithm and we had to adapt the parameters to reflect that difference. We made one parameter vary at a time over 10 executions, in order to maximise the F_1 measure. For each of their parameters we evaluated various parameter value combinations in order to find the parameters that lead to the best average F_1 scores as shown in Tables **Error! Reference source not found.** and **Error! Reference source not found.**.

CR	MR	MN	λ	STH
1.0	**0.15**	1	1000	10
0.9	0.3	20	**500**	**20**
0.6	0.8	50	200	30
0.3	1.0	**80**	50	40

(a) Genetic Algorithm

T_0	ClR	IN
1000	1.00	2000
700	**0.9**	**1000**
400	0.5	700
100	0.3	400
1	0.1	100

(b) Simulated Annealing

Table 3: Parameter values tested for GA and SA(parameter values leading to the highest average score in **bold**

For our ACA, given that an execution of the algorithm is very fast (depending on the parameters the execution takes between 15s to 40s for the first text of the corpus), it was possible to use a simplified simulated annealing approach for the automated estimation of the parameters. Given that the Task 7 corpus provided 5 texts, we decided to use the first text to tune the parameters, and the remaining 4 texts (texts 2 through 5) for the evaluation of the algorithms.

For each parameter combination we ran the algorithm 50 times and considered the means coupled with the p-values from a one-way ANOVA. We still needed to use our *a priori* knowledge to set likely parameter intervals and discrete steps for each of them (although this is a supervised approach to parameter tuning, it does not affect the unsupervised nature of the algorithm itself).

The best parameters we found are:

- For GA: $\lambda = 500, CR = 0.9, MR = 0.15, MN = 80$;
- For SA: $T_0 = 1000, ClR = 0.9, IN = 1000$;
- For ACA: $\omega = 25, E_a = 16, E_{max} = 56, E_0 = 30, \delta_v = 0.9, \delta_\phi = 0.9,$ $L_V = 100$

Where, for GA, λ is the population size, CR is the crossover ratio, MR the mutation ratio and MN the number of mutations.

For SA, T_0 is the initial temperature, ClR the cooling ratio and IN the number of iterations.

For ACA, ω is the life-span of an ant, E_a the quantity of energy an ant take when it passes on a node, E_{max} the maximal quantity of energy an ant can carry, E_0 the initial quantity of energy on each node, δ_v the ratio pheromone evaporation, δ_ϕ the quantity of the definition components deposited into the odour vector and L_V the size of the odour vectors.

7.3 Experimental protocol

The objective of the experiments is to compare the three algorithms according to two criteria: the F_1 score obtained on the Task 7 of Semeval 2007 and the execution time. Furthermore, given that they use the same similarity measure and that it is the computational bottleneck, we also measure the average number of similarity computations between word-senses.

Since the algorithms are stochastic, we need to have a representation of the distribution of solutions as accurate as possible in order to make statistically significant comparisons and thus we ran the algorithms 100 times each.

The first step in the evaluation of the significance of the results is the choice of an appropriate statistical tool. In this case we are using a one-way ANOVA analysis, coupled with a Tuckey' HSD post-hoc pairwise analysis. These two techniques rely on three principal assumptions: independence of the groups, normal distribution of the samples, within group homegeneity of variances.

Since we ran each algorithm 100 times, we have 100 different disambiguations that potentially contain complementary information. Therefore, in order to improve the results we applied a majority vote strategy: we considered 10 result files and for each word selected the sense which was the most frequently selected among the 10 executions. Given that we also want to have 100 results for the vote, instead of making 1000 executions, we took in turns series of 10 successive answer files (0,1,2,3,4,5,6,7,9; then 2,3,4,5,6,7,8,9,10; etc.) and thus from 100 executions we obtained 100 results for the majority voting strategy for all three algorithms. However in the case of the ant colony algorithm we have some additional information that we can use: the level of energy of each nest node. Thus, we are able to use the energy values as weight and perform a weighted voting strategy. Given that we have such values only for ACA, we did not apply this strategy to the two other algorithms.

Thus, we first compare the results of the three algorithms directly, followed by a comparison of the results after a majority voting strategy. We consider both a majority vote and the weighted majority vote for ACA. We also apply an ANOVA significance analysis on the vote results. We use the first sense baseline as the basis of the comparison.

7.4 Quantitative results

In order to check the normality assumption for ANOVA, we computed the correlation between the theoretical (normal distribution) and the empirical quantiles. For all metrics and algorithms there was always a correlation above 0.99. Furthermore we used Levene's variance homogeneity test and found a minimum significance level of 10^{-6} between all algorithms and metrics.

Algorithme	F_1 (%)	σ_{F_1}	$Time(s)$	σ_{Time}	Sim. Eval.	$\sigma(S.Ev.)$
F.S. Baseline	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

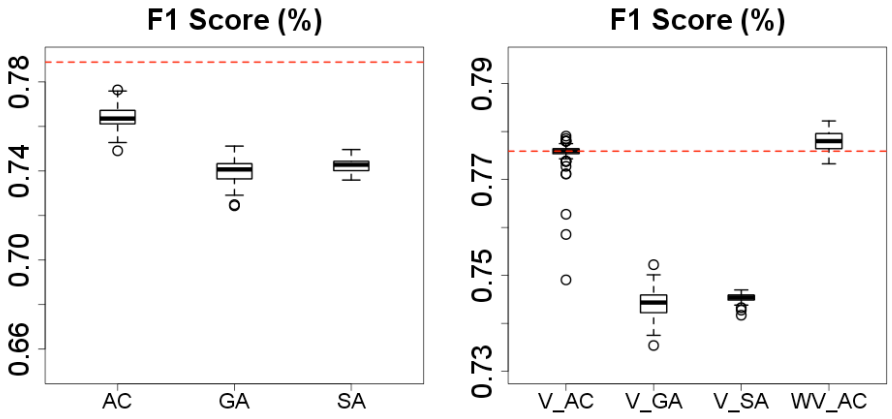
Table 4: Comparison of the F_1 scores, execution times and similarity measure evaluations of the algorithms († $\leftrightarrow p < 0.01$) over texts 2 through 5

Algorithme	F_1 (%)	σ_{F_1} (%)
Weighted Majority Vote	77.79†	0.18
F.S. Baseline	77.59	N/A
Majority Vote ACA	77.50†	0.48
Majority Vote SA	75.18†	0.10
Majority Vote GA	74.53†	0.31

Table 5: Comparison of the F_1 scores of the algorithms after a voting strategy was applied(† $\leftrightarrow p < 0.01$) over texts 2 through 5

Table 4 presents the results for the three algorithms: the F_1 scores, the execution time and the number of evaluations of the similarity measure with their respective standard deviations. Given that the first text was used to train the ACA parameters, in this sections the F_1 scores presented are calculated for the 4 next texts in order to remove any bias. For all three metrics and between all three algorithms, the difference in the means was systematically significant with $p < 0.01$.

Similarly, Table 5 presents the F_1 score for the three algorithms after a majority vote is applied and of ACA with a weighted majority vote. Similarly, a Tuckey HSD post-hoc test indicated a pairwise significance between all the algorithms and vote types with $p < 0.01$.



(a) Normal Execution (b) Vote on the results

Figure 6: Boxplots of the F_1 compared to the first sense baseline (dashed line)

Figures 6 and **Error! Reference source not found.** respectively present boxplots of the distributions of F_1 scores for all three algorithms without and with vote compared to the first sense baseline.

In terms of F_1 score, SA and GA obtain similar results, even though SA is slightly better and shows a lower variability in the score distribution. As for ACA, the scores are on average +1.61% better than SA and +1.76% better than GA, with a variability similar to that of GA. All three algorithms are below the FS baseline, even though the maximum of the ACA distribution is close.

After applying a majority vote on the answer files 10 by 10, for SA and GA there is a slight improvement of the scores ($p < 0.01$), respectively +0.17% and +0.46% and for ACA, there is a larger improvement ($p < 0.01$) of +1.17% (despite some lower-bound outliers). A vote on ACA tends to lead to solutions close to the FS baseline (-0.09%). After the vote, the distribution is practically centred around the latter as far as the score is concerned. After we add to the vote the extra information given by the energy values as weights, the average score increases by 0.29% which makes it higher by 0.20% than the baseline. Furthermore with the weighted majority votes the distribution is more stable as there are no outlier.

In terms of execution times there are huge differences between the algorithms, the slowest being GA, which on average runs over 1.5h (±16 min). SA is much faster and takes on average about 24m (± 2.8m), but still remains much slower that ACA, which converges on average in 65s (± 190ms). As one would expect, the number of evaluations of the similarity measure is directly correlated with execution times of the algorithms (cor_p=0.9969). As for the execution times for the vote strategies, they roughly correspond to the sum of the execution times needed to obtain the 10 result files. As such the proportions between the algorithms are preserved, and while for SA and GA we obtain prohibitively longer times at the scale of roughly 4 hours and 15h respectively, the results for ACA are obtained in a much more reasonable time frame of roughly 11 minutes.

7.5 Comparison to other Task 7 systems

Given that the results of all the other systems are over the 5 texts of the corpus (rather than just 4), we will consider the ACA results for the 5 texts as well, contrarily to the previous section.

Table 6 presents the results for all the systems that initially participated to the workshop as well as the algorithms presented here and other more recent algorithms evaluated on the same task.

Vis-a-vis the original participants, our algorithm is ahead of all other unsupervised systems, and beats the weakest supervised system by getting very close to the first sense baseline. However, most supervised systems are still ahead.

If we add more recent results from the experiments of (Navigli and Pozetto, 2012) using Babelnet, a multilingual database that adds multilingual and monolingual links to WordNet, the Degree algorithm (Navigli and Lapata, 2010) reaches a score -0.63% below ACA (only Wordnet) , followed by Pagerank (Mihalcea et al., 2004) (adapted to BabelNet) -5.43%. When a voting strategy is used, ACA is ahead with 1.75% compared to Degree. With the weighted majority vote, ACA (as presented earlier over 4 texts) is ahead of the baseline by 0.14% and very close behind GPLSI (-0.52%) (Izquierdo et al., 2007), the last supervised system above the baseline. However, it is important to note that when looking at the scores per part of speech, Degree exhibits notably higher results for nouns (85% versus 76.35%), while ACA performs much better for adverbs, adjectives and verbs (83.98%, 82.44%, 74.16%).

As for GA and SA, as we saw before, SA is ahead of GA and compared to the other systems. Furthermore they are both ahead of Pagerank (+1.38% and 1.63%) and behind Degree (-2.78% and -3.3%). After the vote, GA and SA are right ahead of SA with no vote. However GA and SA remain behind Degree (-2.48% and -1.83%).

System	A	P	R	F_1
UoR-SSI‡	100	83.21	83.21	83.21
NUS-PT‡	100	82.5	82.5	82.5
NUS-ML‡	100	81.58	81.58	81.58
LCC-WSD‡	100	81.45	81.45	81.45
GPLSI‡	100	79.55	79.55	79.55
ACA Wght. Maj. Vote (WN)	100	79.03	79.03	79.03
First Sense Baseline	100	78.89	78.89	78.89
ACA Maj. Vote (WN)	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
Simulated Annealing (Vote)	100	75.18	75.18	75.18
Genetic Algorithm(Vote)	100	74.53	74.53	74.53
Simulated Annealing	100	74.23	74.23	74.23
Genetic Algorithm	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
PU-BCD‡	90.1	69.72	62.8	66.08
RACAI-SYNWSD	100	65.71	65.7	65.7
SUSSEX-FR	72.8	71.73	52.23	60.44
USYS‡	95.3	58.73	56.02	57.37
UofL	92.7	52.59	48.74	50.6
SUSSEX-C-WD	72.8	54.54	39.71	45.96
SUSSEX-CR	72.8	54.3	39.53	45.75

Table 6: Comparison with 100% coverage systems evaluated on Semeval2007-Task7 (‡ ↔ supervised system; the systems in *italic* are ulterior to the 2007 Semeval Workshop; the systems in **bold** are the ones presented in this article).

8 Conclusions and perspectives

In this article we have presented three stochastic algorithms for knowledge-based unsupervised Word Sense Disambiguation: a genetic algorithm, a simulated annealing algorithm, and an ant colony algorithm; two from the state of the art, and our own ant colony algorithm. All three algorithms belong to the family of incomplete approaches, which allows us to consider the whole text as the disambiguation context and thus to go further towards ensuring the global coherence of the disambiguation of a whole text. We have estimated the *best* parameter values and then evaluated and compared the three algorithms empirically in terms of the F_1 score on Semeval 2007 Task 7, the execution time as well as the number of evaluation of the similarity measure. We found that the ACA is notably better both in terms of score and execution time. Furthermore, the number of evaluations of the similarity measure are directly correlated with the computation time. Then, we applied a majority vote strategy, which led to only small improments for SA and GA, whilst they were more consequential for ACA. The weighted vote allowed ACA to surpass the first sense baseline and come lowest to the top performing supervised systems.

However, some open ended question remain. The three methods rely on parameters that are not (*a priori*) linguistically grounded and that have an influence both on the results and the computation time. The estimation of the parameters, whether manual or through an automated learning algorithm prevent these algorithms from being entirely unsupervised. However, the degree of supervision remains far below supervised approaches that use training corpora approximately 1000 times larger. Our work is currently focussed on the study of these parameters for ACA. Their values depend on the structure of the text and on its consequences on the environment graph of ACA, as we have outlined in the paper. Moreover, we are now interested in determining the degree to which the similarity measure and the lexical resources it uses influences the parameters. We are currently adapting our $Lesk_{ext}$ to use BabelNet in order to investigate the matter further as well as to enable us to perform Multilingual Word Sense Disambiguation.

Acknowledgements

This paper is a revised and expanded version of a paper entitled *A global ant colony algorithm for Word Sense Disambiguation based on semantic relatedness* presented at the 9th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS'11), that was held from the 6th to the 8th of April 2011, in the city of Salamanca, Spain (Schwab and Guillaume, 2011).

References

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

Banerjee, S. and Pedersen, T. (2002), An adapted lesk algorithm for word sense disambiguation using wordnet, *in* 'CICLing 2002', Mexico City.

Bonabeau, É. and Théraulaz, G. (2000), 'L'intelligence en essaim', *Pour la science* (271), 66–73.

Brin, S. and Page, L. (1998), The anatomy of a large-scale hypertextual web search engine, *in* 'Proceedings of the seventh international conference on World Wide Web 7', WWW7,

Elsevier Science Publishers B. V., Amsterdam, The Netherlands, The Netherlands, pp. 107–117.

Brody, S. and Lapata, M. (2008), Good neighbors make good senses: Exploiting distributional similarity for unsupervised WSD, *in* 'Proceedings of the 22nd International Conference on Computational Linguistics (Coling 2008)', Manchester, UK, pp. 65–72.

Cowie, J., Guthrie, J. and Guthrie, L. (1992), Lexical disambiguation using simulated annealing, *in* 'COLING 1992', Vol. 1, Nantes, France, pp. 359–365.

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

Deerwester, S. C., Dumais, S. T., Landauer, T. K., Furnas, G. W. and Harshman, R. A. (1990), 'Indexing by latent semantic analysis', *Journal of the American Society of Information Science*.

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

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

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

Gale, W., Church, K. and Yarowsky, D. (1992), One sense per discourse, *in* 'Fifth DARPA Speech and Natural Language Workshop', Harriman, New-York, États-Unis, pp. 233–237.

Gelbukh, A., Sidorov, G. and Han, S. Y. (2003), 'Evolutionary approach to natural language wsd through global coherence optimization', *WSEAS Transactions on Communications* **2**(1), 11–19.

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

Hirst, G. and St-Onge, D. D. (1998), '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* pp. 305–332. plus .33em minus .07emEd. MIT Press.

Ide, N. and Véronis, J. (1998), 'Wsd: the state of the art', *Computational Linguistics* **28**(1), 1–41.

Izquierdo, R., Suárez, A. and Rigau, G. (2007), Gplsi: word coarse-grained disambiguation aided by basic level concepts, *in* 'Proceedings of the 4th International Workshop on Semantic Evaluations', SemEval '07, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 157–160.

Kilgarriff, A. and Kilgarriff, A. (1998), Senseval: An exercise in evaluating word sense disambiguation programs, *in* 'First International Conference on Language REsources and Evaluation, LREC', pp. 581–588.

Laarhoven, P. and Aarts, E. (1987), *Simulated annealing: theory and applications*, Mathematics and its applications, D. Reidel.

Lafourcade, M. (2011), 'Lexique et analyse sémantique de textes - structures, acquisitions, calculs, et jeux de mots', HDR de l'Université Montpellier II.

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

Lesk, M. (1986), Automatic sense disambiguation using mrd: how to tell a pine cone from an ice cream cone, *in* 'Proceedings of SIGDOC '86', ACM, New York, NY, USA, pp. 24–26.

Li, Y., Bandar, Z. A. and McLean, D. (2003), 'An approach for measuring semantic similarity between words using multiple information sources', *IEEE Trans. on Knowl. and Data Eng.* **15**(4), 871–882.

Lin, D. (1998), An information-theoretic definition of similarity, *in* 'Proceedings of the Fifteenth International Conference on Machine Learning', ICML '98, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, pp. 296–304.

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

Monmarche, N., Guinand, F. and Siarry, P., eds (2009), *Fourmis Artificielles et Traitement de la Langue Naturelle*, Lavoisier, Prague, Czech Republic.

Navigli, R. (2009), 'Wsd: a survey', *ACM Computing Surveys* **41**(2), 1–69.

Navigli, R. and Lapata, M. (2010), 'An experimental study of graph connectivity for unsupervised word sense disambiguation', *IEEE Trans. Pattern Anal. Mach. Intell.* **32**, 678–692.

Navigli, R. and Pozetto, S. P. (2012), 'Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network'. plus .33em minus .07emhttp://dx.doi.org/10.1016/j.artint.2012.07.004.

Navigli, R., Litkowski, K. C. and Hargraves, O. (2007), Semeval-2007 task 07: Coarse-grained english all-words task, *in* 'SemEval-2007', Prague, Czech Republic, pp. 30–35.

Oliveto, P. S., He, J. and Yao, X. (2007), 'Time complexity of evolutionary algorithms for combinatorial optimization: A decade of results', *International Journal of Automation and Computing* **04**(3), 281–293.

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

Pirró, G. and Euzenat, J. (2010), 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 and B. Glimm, eds, 'The Semantic Web - ISWC 2010', Vol. 6496 of *Lecture Notes in Computer Science*, Springer Berlin / Heidelberg, pp. 615–630.

Ponzetto, S. P. and Navigli, R. (2010), Knowledge-rich word sense disambiguation rivaling supervised systems, *in* 'Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics', pp. 1522–1531.

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

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

Rouquet, D., Falaise, A., Schwab, D., Boitet, C., Bellynck, V., Nguyen, H.-T., Mangeot, M. and 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, Technical report, Agence Nationale de la Recherche.

Schwab, D. (2005*a*), Approche hybride pour la modélisation, la détection et l'exploitation des fonctions lexicales en vue de l'analyse sémantique de texte., PhD thesis, U. Montpellier 2.

Schwab, D. (2005*b*), 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 2.

Schwab, D. and Guillaume, N. (2011), A global ant colony algorithm for word sense disambiguation based on semantic relatedness, *in* '9th International Conference on Practical Applications of Agents and Multi-Agent Systems (PAAMS'11)'.

Schwab, D. and Lafourcade, M. (2007), Lexical functions for ants based semantic analysis, *in* 'ICAI'07- The 2007 International Conference on Artificial Intelligence', Las Vegas, Nevada, USA.

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

Seco, N., Veale, T. and Hayes, J. (2004), An intrinsic information content metric for semantic similarity in wordnet, *in* 'Proceedings of the 16th Eureopean Conference on Artificial Intelligence, ECAI'2004, including Prestigious Applicants of Intelligent Systems, PAIS 2004, Valencia, Spain', pp. 1089–1090.

Silber, H. G. and McCoy, K. F. (2000), Efficient text summarization using lexical chains, *in* 'Proceedings of the 5th international conference on Intelligent user interfaces', IUI '00, ACM, New York, NY, USA, pp. 252–255.

Tchechmedjiev, A. (2012), état de l'art: Mesures de similarité sémantique et algorithmes globaux pour la désambiguïsation lexicale a base de connaissances, *in* 'RECITAL 2012', ATALA, Grenoble.

Wu, Z. and Palmer, M. (1994), Verbs semantics and lexical selection, *in* 'Proceedings of the 32nd annual meeting on Association for Computational Linguistics', ACL '94, Association for Computational Linguistics, Stroudsburg, PA, USA, pp. 133–138.