

Evolutionary Algorithms in Natural Language Processing

Lars Bungum, Björn Gambäck
Department of Computer & Information Science
Norwegian University of Science and Technology
Sem Sælands v. 7-9, 7491
Trondheim, Norway
{larsbun, gamback}@idi.ntnu.no

Abstract—Natural language processing grapples with an ever-changing and moving target. The focus of study, natural language, is natural because it changes, interacts and evolves in various directions. The bio-inspired computational methods described as evolutionary computation create computational models that evolve a population of individuals to find a solution to a given problem. This paper investigates how evolutionary computation has been employed in natural language processing, ranging from efforts to induce grammars to models of language development through parameter optimization and search.

1. Introduction

Natural language processing, the automatic treatment of natural languages (as opposed to constructed) deals with a moving target. The use of language varies across *domains* (application areas and sublanguages) and environmental factors (who speaks to whom and where), as well as being part of a continuing development and change (we even speak a different language from the one spoken in our childhood). Language evolves. Accordingly the idea of *evolutionary* algorithms is a conceptual good fit with language technology as a notion of continuous evolution is in essence a requirement for a fully descriptive account of language. A stale grammar representing a language at a point in time is useful, but not a full description of how this language functions, as new elements are constantly added to it.

This paper will investigate the use of evolutionary algorithms in language technology both from the viewpoint of evolutionary computing itself as well as seen from the language technology side. In doing so, some of the key areas in which evolutionary algorithms have been applied to language processing will be discussed. The purpose of the paper is thus not to give a complete account of all efforts to use evolutionary algorithms in the language processing field, but rather to evaluate the main applications.

A short description of the fundamentals of evolutionary algorithms will be presented in Section 2, before a discussion of their relevance to the processing of natural language in Section 3. A presentation of some notable

applications of evolutionary computation in language processing appears in Section 4, while Section 5 describes some applications in closely related research areas before Section 6 rounds off with a discussion followed by some ideas about future research in Section 7.

2. Evolutionary Computation

In this section, a brief overview of evolutionary computation will be given, presenting some key concepts and terminology that will be used in the later discussion of selected examples.

Evolutionary algorithms are biologically inspired methods to solve problems in Computer Science, but do not aim to hold a 1:1 relationship with nature and biology. However, the methodology assumes a cycle in which some key concepts in Darwinian evolution are retained. De Jong (2006) sums up the key concepts [9]:

- one or more populations of individuals competing for limited resources
- the notion of dynamically changing populations due to the birth and death of individuals
- a concept of fitness which reflects the ability of an individual to survive and reproduce
- a concept of variational inheritance: offspring closely resemble their parents, but are not identical

De Jong goes on to note that although there is no consensus on this, a system would *not* be described as a Darwinian one if one of these components were missing. Even if a simple algorithm as this is no plausible

- 1) $P_0 \leftarrow$ generate initial population of m **individuals**
- 2) Set generational counter $k = 1$
- 3) Evaluate P_0 for **fitness**
- 4) Begin iteration until termination (number of **generations** or termination criteria reached)
 - a) **Select** parents $P_{par} \leftarrow P_{k-1}$
 - b) Get offspring $P_{offsp.}$ by **recombining** parents
 - c) **Mutate** some offspring
 - d) **Select** population to survive unto next generation $P_k \leftarrow P_{k-1} \cup P_{offsp.}$
 - e) Iterate generation counter $k = k + 1$

Fig. 1: Outline of the evolutionary algorithm

model of nature and evolution that takes every biological complexity into account, biology-inspired algorithms can still be used to solve real-world problems. Evolutionary algorithms have been successfully employed in computer science to search, optimization and adaptation to changing environments, as well as being able to model complex behavior in nature. The author points out that the goal and purpose of evolution remains unclear. Where nature's evolution at large is headed is largely a philosophical question; however, the goal of employing an optimization or search algorithm is well-defined. Evolutionary algorithms can be continually expanded and refined through the use of more knowledge from biological systems to close the gap between inspiration source and application, but possibly without fully accounting for every aspect of nature. Despite the discrepancy between algorithms and inspiration, evolutionary algorithms can be used to solve the above-mentioned challenges. In the following, we will visit a few examples.

2.1. Key Concepts

The basic evolutionary algorithm is outlined in Fig. 1

The concepts shown in boldface are key parameters that need to be set when implementing an evolutionary algorithm. Their names underline, and partly explain their role in the biological analogy used as inspiration. However, in an implementation setting, choices will have to be made in how these concepts are to be interpreted.

The individuals of a population can be represented computationally in different ways. Either explicitly, in a format from which the fitness can be calculated directly, or it can be done indirectly via some basic format (like a chromosome) which needs to be **developed** into a form that can be evaluated for fitness. Development, the transition from chromosome to individual, can be done in a number of ways, as will be seen in the examples visited in Section 4. The evaluation of **fitness** is also a function which has to be carefully crafted, as it is not

always obvious what the ideal (fittest) individual would be, the direction in which the individuals are pulled (and certainly no more obvious in biology).

Furthermore, **selection** of individuals must be done according to some method and protocol. **elitism** with a random element is one such selection method, meaning that best individuals (in fitness terms) have the greatest chance of producing offspring into the next generation. How this **recombination** is to be carried out is also a matter of choice, and depends greatly on the representation of the individual: which parts should be recombined, and with what probability. If a direct representation is chosen, there may be severe restrictions on recombination, as the offspring must be in a certain format to be able to have fitness evaluated, whereas a general algorithm can be used for the indirect case. This is also true for **mutation**, a random change in offspring (or on the population as a whole). How much of it to change, and how these parts should change is a matter of choice. Finally, the number of generations need to be chosen as well, and to what degree age is tracked on individuals, affecting their chance of survival.

So even in the basic skeleton of the evolutionary algorithm, many choices must be made in an implementation, and the search for the right setup to solve a task can be arduous. Examples visited in Section 4 show different ways of addressing this. The concepts outlined in Fig. 1 will be used to explain the different applications of evolutionary algorithms in language technology, as the differences between experiments often are related to these choices.

2.1.1. Interaction and co-evolution

Interaction and **co-evolution** are other important concepts, that specify in whether the agents will interact, compete and/or cooperate with each other. Depending on the problems the algorithms are meant to solve, or the phenomena they should model, these elements can be introduced. If individuals are allowed to interact with one another, and this interaction has importance for their fitness scores, interesting behavior on the population can be observed. A much cited example is the evolution of a system of game players, be it checkers, chess or Go [9, pp. 226-228]. Here having the system play against a fixed set of opponents of the same skill would hardly produce a world-class player. But by letting the individuals compete against each other, continuously improving systems can be achieved by competitive co-evolution. Another example where cooperation is central is the creation of a surveillance system where the fitness of each individual is relative to the system as a whole. Co-evolving systems can act as models of *parts of* human behavior where observation of the emergent behavior itself is the point, as well as solving more problems needing interaction as outlined above. Interaction and relative fitness assessments increase complexity in implementations, and

are design choices that are also made as evolutionary algorithms are deployed.

2.2. Terminology

In an account of the history of “evolutionary computation”, De Jong (2006) uses the term as an overarching category for the processes accounted for in the previous section [9]. Evolutionary computation being cyclical algorithms with a biological inspiration. De Jong uses the term “evolutionary algorithms” almost analogously, explaining how they later on split in three directions in the 1970s:

- evolutionary programming
- evolution strategies
- genetic algorithms

The first two are distinguished by the number of offspring created by and added to the initial population. In the third this is parameterized, whereas the first two cases always allow offspring to be produced per population member. More interestingly, De Jong points to the independence between the algorithm and the application in genetic algorithms. This is achieved by having a genotype representation where the individuals are represented as fixed-length binary strings. Using such a representation, the same “genetic” operations as mutation (often the flipping of one bit) and crossover can be applied to problems in many domains, because the actual adaptation to new problems stems from the **development** of the genotype into an individual. Several of the research efforts described in this paper do not follow De Jong’s terminology and let the term *genetic algorithms* describe implementations that use a direct representation of the individual.

3. Natural Language Processing

In this section Natural Language Processing (NLP) will be briefly presented together with an overview of some sub-areas especially relevant to evolutionary computation. NLP is a research field concerned with the interaction between computers and natural (human) language, as spoken and written language bodies are being *processed* for various purposes. The field is situated between Computer Science and Linguistics, and deals with problems ranging from ambiguity resolution both on lexical and syntax level, part-of-speech-tagging (POS-), speech and text segmentation, to syntactic and semantic parsing. The problems have traditionally been solved with either rule-based or data-driven approaches, or in later times combinations of the two.

Major application areas are spelling and grammar checking, machine translation, text summarization, question answering systems, and dialogue systems.

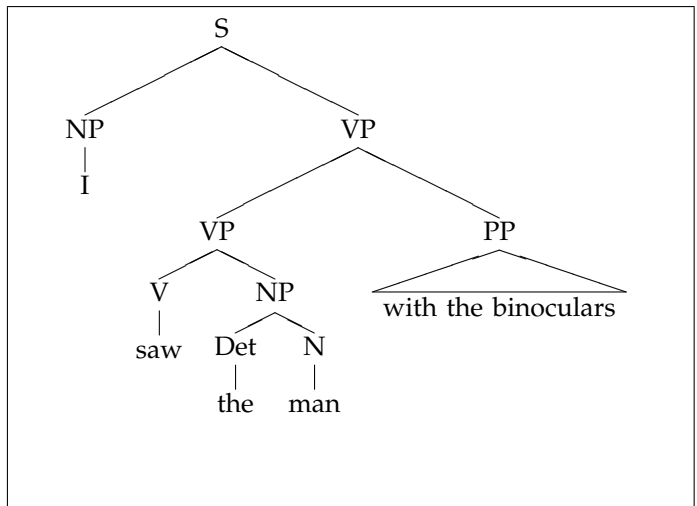


Fig. 2: PP attaching to VP

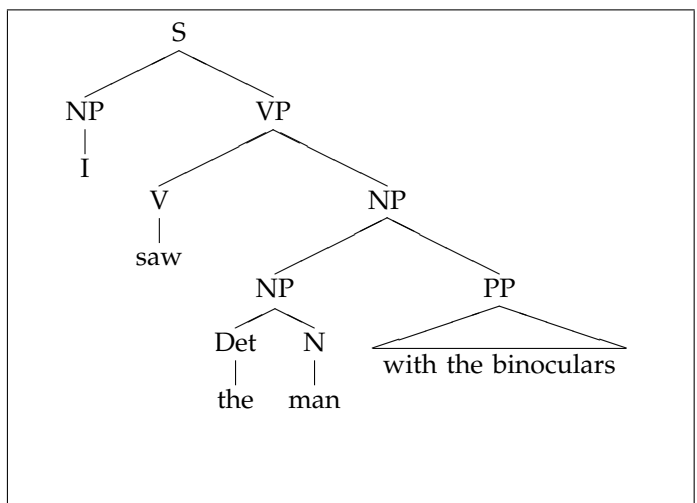


Fig. 3: PP attaching to NP

3.1. Ambiguity Resolution

Because of the inherent ambiguity of natural language, there is a need to perform ambiguity resolution. A much-used example is the parsing of Example (1)

- (1) I saw the man with the binoculars

which is ambiguous in the sense that the preposition phrase (PP) can attach either to the verb phrase (VP) (reading: “I” has binoculars), as seen in Fig. 2 or to the noun phrase (NP) (reading: “the man” has binoculars) as illustrated in Fig. 3.

This classic PP attachment problem increases as the number of prepositional phrases increase where the number of readings are multiplied, as shown in Example (2).

- (2) I saw the man with the binoculars in the forest

- I saw (the man (with the binoculars (in the forest)))
- I saw (the man (with the binoculars) (in the forest))
- I saw (the man) (with the binoculars (in the forest))
- I saw (the man) (with the binoculars) (in the forest)

Fig. 4: Four parses of two PPs

In Example (2) the PP *in the forest* can attach both to the word *binoculars* (formally another NP), or to the overarching VP. The four parses are summed up in Fig. 4

Resolving such structural ambiguities can be done by introducing probabilities to parsers, which will express which parse is the likeliest according to the model. It can also be done by employing well-known machine learning approaches like Maximum Entropy-based rerankers or SVM learners to rank parses based on features extracted from contextual factors. These models introduce a variety of parameters, for some of which there is no analytical method to find the optimal value available.

3.2. Machine Translation

Machine translation (MT) is the transfer of a source language into a target language using a machine. Translation systems can be for specific purposes with a limited amount of legal source sentences, or produce only partly translated output. Hutchins (1995) defines MT as [14]:

The term ‘machine translation’ (MT) refers to computerized systems responsible for the production of translations with or without human assistance.

Hutchins opens for human assistance in his definition, and focuses on the automatic part, that the translation — the transfer between languages — should be done by a machine, excluding dictionary services from the definition, although dictionaries are a necessary prerequisite also for translation.

The term FAHQ(GP)MT is called the holy grail of MT. The acronym stands for *Fully Automated High-Quality General Purpose Machine Translation*, and has been extended from an early formulation (FAHQ(T)) by Bar-Hillel (1953) [2], who later claimed its infeasibility. Because creating a system that can translate any text type for any purpose to another language without human assistance is impossible to attain, the conditions implicit in the acronym have been relaxed in MT systems to date.

3.2.1. Rule-based vs. Data-driven

The trade-off between high coverage (good for many purposes) and high quality determine the set-up of different MT systems. The aforementioned difference between rule-based and data-driven approaches is very visible in MT as well, where rule-based systems generally have better quality for fewer types of text, whereas

data-driven approaches have better coverage for many types, but lower quality output.

A short presentation of the two avenues will be given along these lines. The actual machine translation can be seen as a cascade of natural language processing elements in both approaches, but with different elements. In recent times, combinations of the methodologies are most often employed in so-called hybrid systems.

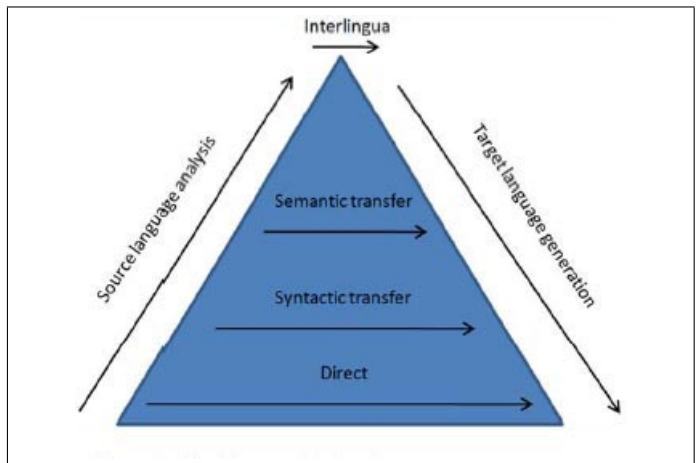


Fig. 5: The Vauquois Triangle

3.2.2. Preprocessing

Machine translation systems depend on some preprocessing of text corpora (text collections) or input strings. Preprocessing typically means POS-tagging, tokenization and lemmatization. Tokenization is determining what the unit of input (a token) to the system should be. Whether *Dr. No* should be one token or two or three, and the treatment of punctuation and special characters is determined here. Lemmatization is the extraction of a *lemma*, the canonical or dictionary form of a word from its realization. The lemma *go* represents the forms *goes* and *went*, for example. The degree to which training data and input strings are preprocessed depends on the characteristics of the systems (type of transfer, for example), but authentic corpora often contain spelling mistakes and many special characters, like sequences of exclamation marks of arbitrary length, that perhaps are better represented with the same token.

3.2.3. Rule-Based Machine Translation

Rule-based translation (RBT) systems have an important role in the history of MT, but are less suitable for evolutionary algorithms, having hand-crafted rules as their basis of operation, and will therefore be sketched briefly. The process of RBT is illustrated by Fig. 5. The input sentence is analyzed, and depending on the depth of analysis, the result is looked up in a table (dictionary if a direct word transfer model is used), and a target language string is generated based on the analyzed

content. The maximum level of analysis is creating an interlingua, a constructed language encompassing all aspects of meaning in the source and target languages.

3.2.4. Statistical Machine Translation

In statistical machine translation (SMT), translation is based on a translation model, a language model and a decoder.

A *translation model* is a statistical model of parallel data (bodies of translated text), predicting the probability of a *unit* (phrase or word) in the target language (TL) being the translation of a unit in the source language (SL). A *language model* is a model of the TL, predicting how likely it is that a given string is a part of the TL. The *decoder* uses the two models to actually find the best TL string provided by the models. The models are trained from parallel data, trying to model a mapping between the languages, based on the knowledge that the SL and TL texts have the same meaning.

3.2.5. Foundations

Formally this is expressed with the fundamental equation of statistical machine translation, formulated as follows:

$$\hat{t} = \arg \max_t P(t|s) \quad (1)$$

The equation expresses \hat{t} , the TL sentence that has the highest probability given the SL input. By means of Bayes rule this can be transformed to

$$\hat{t} = \arg \max_t \frac{P(s|t) * P(t)}{P(s)} \quad (2)$$

The denominator, $P(s)$ is the probability of the input string which is the same for all calculations in the argmax, and can thus be disregarded. The resulting equation corresponds to the equation described at the beginning of this paragraph, where $P(s|t)$ corresponds to the translation model and $P(t)$ to the language model. In a way the problem has been inversed by Bayes; transformed from finding the best TL string given the source, to searching for the best SL string given the target string, multiplied by the probability of this TL string.

3.2.6. Decoding

After having built a translation and a language model, a SL sentence which must be *decoded* can be presented to the system. To be sure of having the string t that is the optimal TL string in this equation, all TL strings would have to be tried to find the one that best translates the source. But the number of possible TL strings t^* is infinite, and cannot be tested. In practice, a decoder needs a formulation of the translation model that is computable, Calculating the product of individual word translation probabilities will not work as a word in a SL might generate two or more words in a TL, or in

some instances nothing at all. The intersection of the translation model and the language model results in a huge search space, which has to be reduced by heuristics to be tractable. Knight (1999) has shown that decoding is NP-complete [16]. The combination of possible translation model output with language model probabilities can be represented as a large graph, and graph theoretic methods like A*-search and beam search can be used to find a TL string.

3.2.7. Alignment

Alignment is introduced to statistical machine translation models as it provides information about how many TL words a SL word will generate. The IBM models 1-5 [5], are all based on models of alignment, with the later models bringing in more variables like fertility (words generating more words in the TL) and distortion (words changing places in the TL sentence). Depending on the model, Eq. 2 is approximated with products of word probabilities factoring in variables like alignments and fertility. An alignment is an account of what words in a SL string correspond to what words in a TL string. When a parallel corpus is aligned, statistical data can be collected.

In Example (3), words 1, 2 and 3 from the SL correspond to words 1, 2 and 3 in the TL.

(3) I am here
Jeg er her

Alignments between words can be formally included in the translation model equation as

$$P(s|t) = \sum_{a \in A} P(s, a|t) \quad (3)$$

replacing the translation model in Eq. 2.

where A is the set of all alignments. Summing over all possible alignments gets computationally costly as the lengths of the sentences grow. Finding the best alignment is a good approximation to the translation probability. Both phrase- and word-based models use alignments to model translation probabilities.

3.3. Grammar Induction

A grammar of a language is a set of structural rules that determine the relation between sentences, phrases and words in a language. In this section we will discuss efforts to automatically induce such grammars from data.

Adriaans and van Zaanen (2007) see computational grammar induction as the identification of an infinite structural description on the basis of a finite number of examples [1]. This contrasts with grammar engineering where a grammar is written manually by experts, analogous with the rule-based approaches to machine

translation discussed in Section 3.2. The authors describe different approaches to the problem, revisiting philosophical deliberations about the nature of language (whether language is innate in each person or learned from scratch), highlighting some known algorithms for computational grammar induction.

Adriaans and van Zaanen argue the existence of three independent research avenues into this problem. First a recursive-theoretic approach where a *learner* asks questions that are truthfully answered by an oracle, which will result in a description of the grammar by the learner. Second, algorithms for the unsupervised identification of a grammar from a text corpus, that try to establish context-free grammar rules using clustering (grouping chunks of words). The use of genetic algorithms in grammar induction is part of this second research tradition. Third, probabilistic methods, that are supervised methods learning a formal grammar from annotated data. Probabilistic parsers by Charniak (1996) [7] and Collins, (2003) [8] are examples of this. An experiment tying languages together based on phylogenies is commented briefly in Section 5.2.

4. Evolutionary Algorithms Applied

Following the discussion of some selected subproblems of natural language processing, this section will provide a few examples of the application of evolutionary algorithms. The efforts are largely related to parameter optimization, but a renewed interest in historical linguistics from the computational side may open for further expansion of the common ground shared by language technology and the evolutionary computing community. A dawning field of evolutionary linguistics encompasses many of the principles from evolutionary algorithms in its efforts to understand language development through phylogenous reasoning and experiments with interacting agents.

4.1. Text summarization

The essence of a document may well be contained in a good summary, being a motivation for automatic summarization systems. The availability of such software may make information more available in principle, and make the workday simpler for professionals overloaded with information in particular. Litvak et al. (2010) use a genetic algorithm to explore a large search space where an optimal linear combination of the statistical sentence scoring measures for use in automatic summarization [20]. Each of the sentence measures rank the sentences after how likely they are to contain the most important parts of the document according to *its* measure.

Litvak et al. aimed to create “language-independent” methods, which means that the summarization engine could not be based on any language-specific knowledge.

The approach was based on a linear optimization of 31 sentence ranking measures. A genetic algorithm was used to find the optimal ways of the 31 methods. The method was tested on English and Hebrew, and showed performance improving on the state-of-the-art in multilingual summarization. The sentence scoring measures were based on either vectors, or graph representations of a document, in turn based on word segmentation. These measures include proximity to the beginning and end of the document, the number of words and characters, and similarity with the title.

4.1.1. Genetic Algorithm Implementation

The evolutionary algorithm setup follows the principles outlined in Section 2, with the use of a direct representation of the individual [20]. An individual here is a set of weights for each ranking measure, stored in a vector. The direct representation means that the genetic operators have to take this representation (its length) into account when doing crossover and mutation, as opposed to generalized genetic operators that could have been used on an indirect representation as binary numbers, transformed into a weight vector for fitness evaluation.

Formally the weight vector consists of the weights for each of the methods used: $\vec{v} = \langle w_1, \dots, w_D \rangle$. This is a direct representation, making no genotype to phenotype distinction, and the fitness evaluations can be performed, calculating fitness directly on the vector.

The selection method used is *elitism*, retaining the best fit of a 500 member population, before performing crossover creating a weighted vector between the two selected parent vectors $\vec{v} = \lambda * \vec{v}_1 + (1 - \lambda) * \vec{v}_2$. Mutation is done on 3% of the individuals changed by a uniformly randomized factor in the range of $[-0.3, 0.3]$. The algorithm stops when it converges with improvements less than a pre-defined threshold.

4.1.2. Evaluation and Results

The experiments were evaluated with the standard ROUGE-measure for text summarization [19], and measured against TextRank, the best known multilingual summarizer and Microsoft Word’s autosummarize function. As further baselines, a summary compiled from only the starting sentences was created, as well as the two best of the 31 measures used in combination. The combined system created with a genetic algorithm performed better than all of the above metrics.

4.2. Language Learning Using Genetic Algorithms

Smith and Witten (1995) employ a genetic algorithm in their early work on grammar induction [25].

They build on works by Koza (1992) to detect exons of a fixed length (5) [17]. Koza used a grammar represented as logical S-expressions, using the operators

AND, NOT and OR, as in for example (AND (OR a the) (OR cat dog)) to generate the strings *a dog/the dog/a cat/the cat*. A subset of context-free grammars can be translated to this form, so Smith and Witten had to avoid recursive production rules in their work. Such rules are exemplified by $NP \rightarrow NP, PP$ as shown in Fig. 3, that lets a rule invoke itself, opening for sentences of infinite length.

4.2.1. Genetic Algorithm Implementation

The genotype — or *chromosome* in the terminology of Smith and Witten — is a context-free grammar in the S-expression form, whose fitness can be evaluated on its merits of “covering” a training set of sample strings. Context-free grammars can be rewritten into Chomsky Normal Form (CNF), which in turn can be translated to S-expression, which loses some readability but has computational advantages. Covering here means the ability to parse, accept or generate the strings based on this grammar. The object of the task is to induce a grammar which best accounts for the training set, and evaluate this on a test set.

Selection is carried out in inverse proportion to the grammar’s size. Smaller (termed parsimonious) grammars have a greater chance of being selected. The reproduction itself is done by combining random nodes from one of the grammars. The S-expression grammar can be drawn in a tree form, and would mean exchanging one branch for another, as illustrated by Figures 6 and 7 where the bottom right nodes can be exchanged for one another, replacing (AND a cat) with (OR the cat).

Mutation is also implemented by randomly choosing one grammar (individual), this time in proportion to the grammar size, meaning a larger grammar is more likely to be selected for mutation. The mutation takes the form of exchanging one of the nodes in the grammar with the complement, if it is headed by an operator (exchanging AND for OR and vice versa). If it is a leaf node, a randomly selected terminal symbol (word) from the grammar is selected.

For each cycle of the genetic algorithm, the grammars that were able to parse the largest number of test sentences were able to persist, alongside any grammars that were able to parse a similar number of sentences, meaning that the number of grammars could grow indefinitely.

4.2.2. Evaluation and Results

Three categories of experiments were conducted to test the behavior of the genetic algorithm. First the algorithms were qualitatively evaluated on their behavior on some simple English sentences, testing how adding more sentences to the test set and increasing the number of generations would help make the induced grammar account for more phenomena in the language samples. Second, robustness was explored by checking if the order

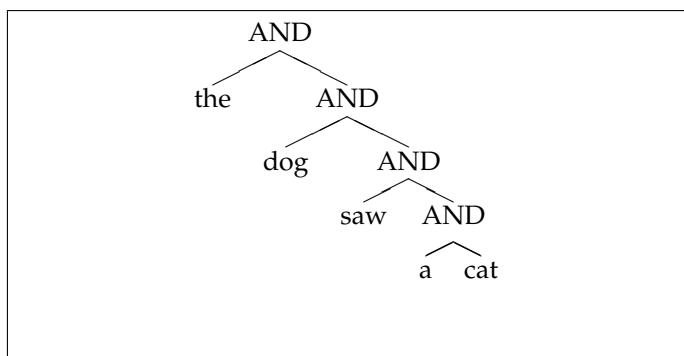


Fig. 6: S-expression 1

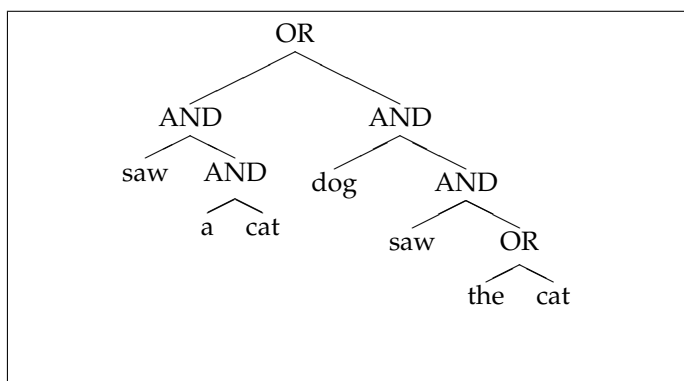


Fig. 7: S-expression 2

of adding sentences to the training set would influence the grammar’s ability to account for language phenomena. Third, the effectiveness of the genetic algorithm was discussed. CNF-grammars can be converted into S-expressions, but the reverse is also possible. When reversing an induced grammar back to CNF form, it was noted that it contained extraneous nodes compared to an engineered grammar for the same language fragment, which the authors attributed to a possible over-use of mutation. These results seem mostly interesting in helping develop adequate experiments, and are hard to compare to other grammar induction attempts. As an early work in the field, the qualitatively extracted lessons from the experiments still hold value.

4.3. Evolutionary Computing as a Tool for Grammar Development

De Pauw (2003) presents a larger framework (the Grammar Evolution framework GRAEL) which also employs evolutionary computing to induce a grammar from an annotated corpus [11]. Although he does not present the experimental details of the evolutionary computation, most notably the parameters, the approach is outlined by separating the grammar induction effort into three;

First, a system for bootstrapping a grammar from an annotated corpus through language games, second a

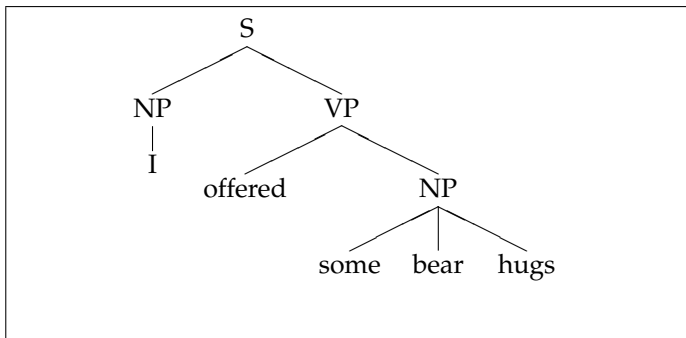


Fig. 8: ag1's treebank sentence

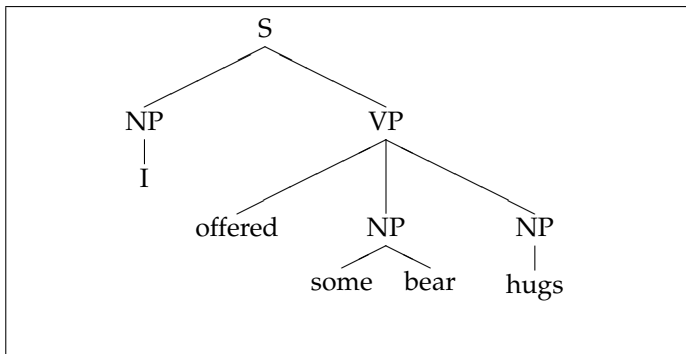


Fig. 9: ag2's attempted parse

method to allow for mutation, and third a new system for completely unsupervised grammar induction. The latter is presented in a more scarce form and left out of the discussion. The starting point of the experiments is a collection of syntactic trees, such as those shown above, which are randomly distributed out on a fixed number of *agents*. From this initial treebank, a probability distribution of the various linguistic phenomena can be acquired by counting frequencies of occurrence (of, say, the rule $S \rightarrow NP, VP$).

4.3.1. Language Games

In a *language game*, the first agent (ag1) will present a sentence, for which the other agent (ag2) then will present a parse. If this is not correct, ag1 will try to help ag2 by providing it with the part it got wrong. This is illustrated in Figures 8 and 9.

As the VP part is wrong in ag2's parse, ag1 will provide this part to ag2 to include in its treebank so it knows better how to deal with such a situation. If the sentence then is correctly parsed, the agents move on to the next sentence in ag1's treebank. This will lead to the treebanks of the various agents growing a lot as they run across unseen phenomena. Therefore, new generations are introduced, and agents survive depending on a fitness measure. The fitness of the grammars depends on parsing accuracy, but also on speed and effectiveness.

4.3.2. Introduction of Mutation Operator

In the second version of the framework, a mutation operator is used in an attempt to discover linguistic phenomena not present in the training set, but possibly in a held-out test set. By mutating the grammar, it can explain language phenomena which it has never seen in its training material.

The mutation itself is done by deletion, addition and replacement of nodes in the tree-structure, equipping agents with unseen grammatical data, from which good grammatical rules (as will be discovered through games) can be inferred. If the rules brought on by mutation are in fact able to cover sentences from the test corpus they will be retained. De Pauw notes that this is a stretch of terminology, as the mutation is not performed on a chromosome level (indirect representation), but on the actual tree structures themselves.

4.3.3. Evaluation and Results

The GRAEL experiments were evaluated on their ability to parse datasets from the Penn Treebank [21], and the smaller ATIS corpus. The PMPG parser [10] is used to parse the datasets with the grammars from the experiments. PMPG is a CKY-based [6] parser with a post-parsing parse forest re-ranker based on probabilistic information.

In the GRAEL-1 (first) experiment, a grammar is induced from the training set directly and used to evaluate the test set. This parsing accuracy is used as baseline. The same training set is the distributed over a number of GRAEL-1 agents, that will interact in language games, resulting in another grammar. A number of experiments were conducted exploring generation methods, fitness functions and termination criteria. A selection of these results were presented, and the GRAEL-1-evolved grammars were able to improve on the baseline, with a greater number of agents generating better results.

The GRAEL-2 experiments involving a mutation operator to discover grammar rules unseen in the training set were not able to outperform the accuracy of GRAEL-1.

4.4. Alignments in Statistical Machine Translation

Rodriguez et al. (2008) tested several implementations of evolutionary algorithms to find the alignment between two sentences for use in statistical machine translation [24]. The role of alignment models in statistical machine was discussed in Section 3.2.7. The authors argue the importance of improving the alignment models, and not just the decoding, or search for a translation.

Rodriguez et al. studied the use of evolutionary algorithms to find the optimal alignment between two sentences, comparing several evolutionary algorithms with each other as well as with alignments created by human experts. The agreement among these were not

full, so the scoring was divided between a full score for attaining an alignment about which there was full agreement within the expert group, and a partial score for developing an alignment one or more of the experts had approved.

Given a source sentence s and a target sentence t , searching for the best alignment between both sentences can be defined with Eq. 4.

$$\hat{a} = \arg \max_a P(a|s, t) = \arg \max_a P(s, a|t) \quad (4)$$

The similarity between this equation and the IBM models means that the IBM word models [5] can be used to evaluate the fitness of the alignments in the search space, as they provide ways of calculating the probabilities on the right-hand side of Eq. 4.

4.4.1. Evolutionary Algorithm Implementation

The individuals in these EA experiments were alignments represented directly by a vector $\mathbf{a} = \mathbf{a}_1, \dots, \mathbf{a}_J$ where J represents the length of the source sentence s . The value of the members of the vector can be 0 in case the word in the target language sentence is not aligned to any word, or any of the word positions in the source language. The search space to explore is the number of possible alignments, and the objective is to find the best one.

Though the representation of the individuals is quite easy, and this simple representation was retained also for fitness evaluation, the evaluation uses the IBM model 4 way of calculating a translation model probability. This model takes fertility, the amount of words one SL word will generate in the TL into account. When each word can generate potentially many words in the TL, the number of possible alignments increases greatly and can not be easily computed. The authors present evolutionary algorithms as a plausible alternative for searching for the optimal alignment to the greedy methods used to date.

Rodriguez et al. (2008) [24] explored three different evolutionary algorithms. First a basic genetic algorithm with a simple cycle of an initial population that after selection propagates through crossover, then mutates and retains a number of the population. The genetic algorithm used elitism for replacement (the n fittest individuals survive) with random mutation (0.01 probability) and one-point crossover (0.9 probability). The individuals are solutions to the problem, a *direct* representation. In addition to this traditional genetic algorithm, two types of estimation of distribution algorithms were used.

The high number of parameters in such algorithms (in itself a vast scope to explore) motivated the authors to explore estimation of distribution algorithms [18]. These algorithms differ from normal genetic after the populations are initialized. At this point, a group of individuals are selected from the initial population. A

probabilistic model is built from the selected group, and another group of individuals is sampled from this model and evaluated for fitness. The next generation is then a selection of the union of these two groups. This process replaces crossover and mutation as described in the basic genetic algorithm. Finding the joint probability distribution over alignments is a difficult calculation, as the number of possible alignments is very high with the introduction of fertility.

Two versions of estimation of distribution algorithms were explored, univariate marginal distribution algorithm (UMDA) and population-based incremental learning (PBIL).

UMDA approach is computationally simple, as the n -dimensional probability distribution is factorized as the product of n univariate distributions, as shown in Eq. 5.

$$Pr(X_1, \dots, X_n) = \prod_{i=1}^n Pr(X_i) \quad (5)$$

When independence between the variables is assumed, they can be factorized as the product.

The PBIL approach also assumes marginal independence, but involves a Hebbian-like learning rule moving the probabilistic model towards the best solution in the current population at each time, updating each component in the model, $Pr(X_i)$, during each iteration.

4.4.2. Evaluation and Results

Evaluating their approach, both of the EDA approaches outperformed the simple GA baseline (although the parameter space here was not fully explored), and performed competitively against state-of-the-art aligners like the GIZA++ [22] alignment method. The expert-created alignments were sorted into *Sure* and *Possible* groups depending on the whether there was full inter-annotator agreement or not. Precision, recall, F-score, and a measure joining the two groups were used in evaluation. As the evolutionary algorithms are non-deterministic, the scores were averages over 30 runs. Rodriguez et al. (2008) noted that while the UMDA approach is computationally much faster, it was able to perform better than the GE and PBIL, although the difference decreased as the number of iterations increased at a computational cost [24]. Therefore the actual advantage of UMDA is speed in their findings.

The authors also conducted additional experiments splitting up the model errors from the search errors, lending support to their claim that more emphasis should be put on improving the models, as the search errors were few.

5. Related Research

Although evolutionary algorithms in the way described in Section 2 are not used directly, new computational historical linguistic efforts draw similar inspiration

from evolution as evolutionary algorithms. The advances in computational techniques both inspired by biology and machine learning in general has increased interest for computational experiments in historical linguistics.

A language phylogeny is a chart of the historical evolution of natural language. It will say that Norwegian and English belong to the North Germanic tree, whereas English and Dutch belong to the West Germanic language family, and these groups are again Germanic.

5.1. Phylogenies for Cognate Word Identification

Hall and Klein (2010) present language phylogenies as a means to automatically identify *cognate* words from unaligned word list, given only the known family tree of languages [13]. Cognates are words that are realized as descendants of the same ancestor in an ancestor language, such as the English word *night*, and the Norwegian word *natt* or Slovak word *noc* having a similar origin. This experiment is a study in computational etymology, modeling how languages develop over time.

The generative model is based on three sub-processes, *survival*, *evolution*, and *alignment*. Survival is a parameterized probability of a cognate actually getting a realization in the language in the following generation. Evolution is the process by which the words change as they drift down the language family tree, conducting the most explicit modeling of the etymology of the word realizations. Alignment of the word lists used as input actually accounts for which words belong to the same cognate groups, by keeping track of their positions in the word lists.

5.1.1. Results

The experiments begin by providing a number of cognate groups, going down the phylogeny tree through survival and evolution, and finally align them to words in the word lists if they are still present. The authors report good results for identifying cognate groups, 90.6% identification accuracy for cognate groups that have realizations in all leaf languages, and slightly lower for the partial ones, both beating the baseline.

5.2. Phylogenies for Grammar Induction

In an experiment inducing unsupervised dependency grammars for multiple languages without parallel texts, Berg-Kirkpatrick and Klein (2010) instead use a probabilistic prior to couple the languages at a parameter level [3]. The authors show how languages (phylogenetic) relations help induce good grammars.

An unsupervised approach without parallel (translated) text cannot employ multilingual constraints as the models do not know what bodies of text are translations of each other. Instead, the relations between languages are captured as *priors* in the probabilistic models. This

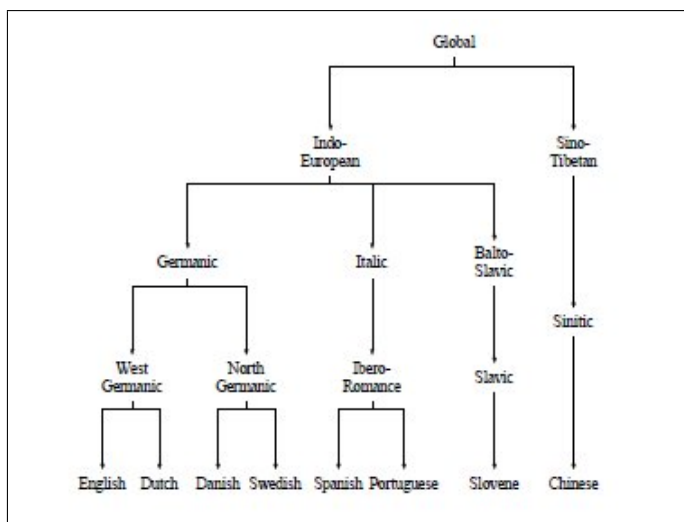


Fig. 10: Language phylogeny used in experiments

means that the set of parameters to optimize when inducing the dependency grammar depends on the set of child languages belonging to the parent node at a given place in a phylogenetic tree structure.

The authors compared many models, pairing the languages English, Dutch, Danish, Norwegian, Spanish, Portuguese, Slovene and Chinese in one pool across families, then doing pairwise comparisons, and also an experiment using a phylogeny with three layers. The relations used in the experiments are shown in Fig. 10.

5.2.1. Results

The experiments were conducted in three main versions. First languages were only coupled, a small reduction (5.6%) of error compared to the monolingual baseline was attained. Second a flat global model, treating all listed languages as equally related reduced error by 10%, and finally a phylogeny capturing intra- and interfamily relations reduced error by 21.1%.

5.3. Evolutionary Linguistics and Language Games

5.3.1. Background

Bleys and Steels (2009) describe a language game-based experiment in which the establishment, interaction and evolution of language strategies are explored [4]. A language strategy can here be exemplified with the English language going from being brightness-based, to being hue-based. The word “yellow” in Old English meant “to shine” and was used to describe brightness, but changed to a hue-based strategy in Middle English to describe yolk or colored paper.

5.3.2. Language games

The authors also did experiments with *embodiment*, with actual robots perceiving color in their environments.

Importantly, the agents are independent, autonomous pieces of soft- or hardware that cannot read each other's memory. The language game is played with two agents randomly chosen and presented to two new, distinguishable color samples. One agent is chosen as a speaker, which again chooses one sample as the *topic*. The topic is then presented to the other agent by a word, selected based on the Euclidean distance from the templates the agent already has for colors. If the hearer correctly identifies the right sample the game is a success, otherwise the right answer is presented by the speaker. After running many generations of language games, the agents converge to a common vocabulary for describing the colors, representing some language strategy.

5.3.3. Relation to Evolutionary Computing

The experiments do not follow the evolutionary computation cycle described in Section 2, but still uses elements of it, also finding inspiration in biology and evolution. Although no notion of generations exist, the number of language games per agent is parameterized. Each language strategy is given a fitness score based in its success in communication, and the individual agents keep track of which strategy was used to acquire new words by tagging their own vocabulary with the employed strategy.

The experiments were able to show that the agents shifted from a simple strategy to a more elaborate one as they were exposed to more color nuances. The experiments were thereby successful in modeling such a shift evidenced in English language history.

6. Discussion

As Section 4 exemplifies, evolutionary computing being a well-known method parameter optimization has found its place also in natural language processing. While evolutionary algorithms are well-established parameter optimization and search methods, evolutionary algorithms involving interaction can produce interesting results in the structures that emerge from agent interaction. The cited examples in grammar induction and language strategy evolution show this.

In this article we have discussed a selection of evolutionary computing examples within NLP research. Text summarization has been optimized with evolutionary algorithms in Section 4.1, grammars have been induced in Section 4.3 and alignment models for statistical machine translation have been created in Section 4.4, as well as related research in language evolution and etymology being discussed in Section 5.

Many NLP tasks involve fine-tuning of parameters and search, and evolutionary algorithms have been used for this in many other projects than those we have performed a detailed discussion of. For example in machine translation where Sofianopoulos and Tambouratzis

(2010) used a genetic algorithm to optimize parameters for a hybrid MT system [26] (explained in Section 3.2), while Otto and Rojas (2007) used evolutionary algorithms for the decoding [23]. Decadt et al. (2004) used genetic algorithms in word-sense disambiguation for the optimization of classifiers parameters in an unsupervised approach [12], and Johansson (2003) introduced an indirect representation of words in a genetic algorithm to find features for a nearest-neighbor mechanism in word classification (clustering) [15].

7. Conclusions and Further Research

Evolutionary algorithms work well for parameter optimization, but only make up a subset of the search algorithms for a large solution space, much like search with a randomized component. For feature-based classification tasks, other techniques such as Maximum Entropy and SVM-based learners achieve state-of-the-art performance. Even if evolutionary algorithms can be competitive, they do not present a significant improvement over other systems.

More interestingly, evolutionary computing involving interaction of agents can model complex behavior in language, such as the development of language strategies, or the induction of grammars. The set of legal sentences in natural language is infinite, whereas the set of examples to learn from is finite. How can the description of an infinite structure with finite means be valid? This philosophical problem has been addressed in many forms and is related to the 17th century discussions between Hume and Descartes about rationalism vs. empiricism, and also to 20th century discussions about universal grammar (the idea that language is innate and similarly represented in all people) often associated with Chomsky.

Evolutionary computing is one possible way of addressing, or perhaps adhering to this dilemma. Grammars that are induced from agent interactions learn and adapt to new samples, and can in principle discover grammatical structures (utterances) that have not yet been uttered. On a more concrete level, there is a need in most NLP application areas to be able to dynamically adapt to new language domains (sublanguages), and evolutionary algorithms is one way of doing this. It would be interesting to see an evolutionary algorithm applied to rebuilding and reconfiguring a translation model in statistical machine translation dynamically as new training instances and corrections are being provided by the user.

In facilitating the emergence of structures from data, and the modeling of language development and etymology, evolutionary computation has shown a lot of promise, helped by the present-day increase in available data and computing power. As computer resources

grow, more agents and more generations can be introduced. As computational linguistics plays a bigger part in historical linguistics, evolutionary algorithms should have a part in this.

8. Acknowledgements

This work was funded by the PRESEMT project¹ sponsored by the European Commission as part of the Information Society Technologies (IST) programme under EC grant number FP7-ICT-4-248307.

We would like to thank Keith Downing for providing feedback and explanations improving this article.

References

- [1] P. W. Adriaans and M. M. van Zaanen. Computational Grammar Induction for Linguists. *Grammars*, 7:57–68, 2004. Special issue with the theme “Grammar Induction”.
- [2] Y. Bar-Hillel. A Quasi-Arithmetical Notation for Syntactic Description. *Language*, 29:47–58, 1953. Reprinted in Y. Bar-Hillel. (1964). *Language and Information: Selected Essays on their Theory and Application*, Addison-Wesley 1964, 61–74.
- [3] T. Berg-Kirkpatrick and D. Klein. Phylogenetic Grammar Induction. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1288–1297, Uppsala, Sweden, July 2010. Association for Computational Linguistics.
- [4] J. Bleys and L. Steels. Linguistic Selection of Language Strategies: A Case Study for Color. In *Proceedings of the 10th European Conference on Artificial Life (ECAL2009)*, LNCS, Budapest, Hungary, 2009. Springer.
- [5] P. F. Brown, V. J. Pietra, S. A. D. Pietra, and R. L. Mercer. The Mathematics of Statistical Machine Translation: Parameter Estimation. *Computational Linguistics*, 19:263–311, 1993.
- [6] J.-C. Chappelier and M. Rajman. A Generalized CYK Algorithm for Parsing Stochastic CFG. In *Proceedings of the 1st Workshop on Tabulation in Parsing and Deduction*, pages 133–137, 1998.
- [7] E. Charniak. *Statistical Language Learning (Language, Speech, and Communication)*. The MIT Press, 1996.
- [8] M. Collins. Head-Driven Statistical Models for Natural Language Parsing. *Computational Linguistics*, 29(4):589–637, 2003.
- [9] K. A. De Jong. *Evolutionary Computation: A Unified Approach*. MIT Press, 2006.
- [10] G. De Pauw. Aspects of Pattern-matching in Data-Oriented Parsing. In *Proceedings of the 18th International Conference on Computational linguistics*, pages 236–242, Morristown, NJ, USA, 2000. Association for Computational Linguistics.
- [11] G. De Pauw. Evolutionary Computing as a Tool for Grammar Development. In *GECCO’03: Proceedings of the 2003 International Conference on Genetic and evolutionary computation*, pages 549–560, Berlin, Heidelberg, 2003. Springer-Verlag.
- [12] B. Decadt, V. Hoste, W. Daelemans, and A. V. den Bosch. GAMBL, Genetic Algorithm Optimization of Memory-Based WSD. In R. Mihalcea and P. Edmonds, editors, *Proceedings of the Third International Workshop on the Evaluation of Systems for the Semantic Analysis of Text (Senseval-3)*, pages 108–112, 2004.
- [13] D. Hall and D. Klein. Finding Cognate Groups Using Phylogenies. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1030–1039, Uppsala, Sweden, July 2010. Association for Computational Linguistics.
- [14] W. J. Hutchins. Machine Translation: A Brief History. In *Concise History of the Language Sciences: from the Sumerians to the Cognitivists*, pages 431–445. Pergamon Press, 1995.
- [15] C. Johansson. Searching for Features Using a Genetic Algorithm. In B. Tessem, P. Ala-Siuru, P. Doherty, and B. Mayoh, editors, *Proceedings of the 8th Scandinavian Conference on Artificial Intelligence (SCAI’03)*, Frontiers in Artificial Intelligence and Applications, pages 119–130, Bergen, Norway, 2003. IOS Press.
- [16] K. Knight. Decoding Complexity in Word-replacement Translation Models. *Computational Linguistics*, 25(4):607–615, 1999.
- [17] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, Cambridge, MA, USA, 1992.
- [18] P. Larranaga and J. A. Lozano. *Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation (Genetic Algorithms and Evolutionary Computation)*. Springer, 2001.
- [19] C.-Y. Lin. ROUGE: A Package for Automatic Evaluation of Summaries. In *Proc. ACL workshop on Text Summarization Branches Out*, page 10, 2004.
- [20] M. Litvak, M. Last, and M. Friedman. A New Approach to Improving Multilingual Summarization Using a Genetic Algorithm. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 927–936, Uppsala, Sweden, July 2010. Association for Computational Linguistics.
- [21] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a Large Annotated Corpus of English: The Penn Treebank. *Computational Linguistics*, 19(2):313–330, 1993.
- [22] F. J. Och and H. Ney. A Systematic Comparison of Various Statistical Alignment Models. *Computational Linguistics*, 29(1):19–51, 2003.
- [23] E. Otto and M. C. R. Rojas. EDA: An Evolutionary Decoding Algorithm for Statistical Machine Translation. *Applied Artificial Intelligence*, 21(7):605–621, 2007.
- [24] L. Rodríguez, I. García-Varea, and J. A. Gámez. On the Application of Different Evolutionary Algorithms to the Alignment Problem in Statistical Machine Translation. *Neurocomputing*, 71(4-6):755–765, 2008.
- [25] T. C. Smith and I. H. Witten. Learning Language Using Genetic Algorithms. In *Connectionist, Statistical, and Symbolic Approaches to Learning for Natural Language Processing, volume 1040 of LNAI*, pages 132–145. Springer, 1995.
- [26] S. Sofianopoulos and G. Tambouratzis. Multi-Objective Optimization of Real-valued Parameters of a Hybrid MT System Using Genetic Algorithms. *Pattern Recognition Letters*, 31(12):1672–1682, 2010.

¹<http://www.presemt.eu>