Studying the Advantages of a Messy Evolutionary Algorithm for Natural Language Tagging *

Lourdes Araujo

Dpto. Sistemas Informáticos y Programación.
Universidad Complutense de Madrid.
lurdes@sip.ucm.es

Abstract. The process of labeling each word in a sentence with one of its lexical categories (noun, verb, etc) is called tagging and is a key step in parsing and many other language processing and generation applications. Automatic lexical taggers are usually based on statistical methods, such as Hidden Markov Models, which works with information extracted from large tagged available corpora. This information consists of the frequencies of the *contexts* of the words, that is, of the sequence of their neighbouring tags. Thus, these methods rely on the assumption that the tag of a word only depends on its surrounding tags. This work proposes the use of a Messy Evolutionary Algorithm to investigate the validity of this assumption. This algorithm is an extension of the fast messy genetic algorithms, a variety of Genetic Algorithms that improve the survival of high quality partial solutions or *building blocks*. Messy GAs do not require all genes to be present in the chromosomes and they may also appear more than one time. This allows us to study the kind of building blocks that arise, thus obtaining information of possible relationships between the tag of a word and other tags corresponding to any position in the sentence. The paper describes the design of a messy evolutionary algorithm for the tagging problem and a number of experiments on the performance of the system and the parameters of the algorithm.

1 Introduction

The process of labeling each word in a sentence of a text with its lexical category (noun, verb, etc) is called *tagging* and is a key step in the parsing process and many other natural language processing and generation applications: machine translation, information retrieval, speech recognition and generation, etc.

A word may have more than one possible tag (lexical ambiguity), and thus, disambiguation methods are required to proceed with the tagging. There are different approaches to automatic tagging based on statistical information, that use large amount of data to establish the probabilities of the tag assignments. Most of them are based on Hidden Markov Models (HMMs) and variants [11, 4,

* Supported by project PR1/03-11588.

13, 3] and neither require knowledge of the rules of the language nor try to deduce them. Others are rule-based approaches that apply language rules to improve the accuracy of the tagging. The Brill system [2] extracts these rules from a training corpus, obtaining competitive performance with stochastic taggers.

Evolutionary algorithms present an appealing tradeoff between efficiency and generality and for this reason they have been extensively applied to complex optimization problems. They have previously been applied to tagging [1]. In the evolutionary tagger, individuals are sequences of tags assigned to the words of a sentence. This model is a variant of the HMM in which disambiguation is introduced by assigning different probabilities to a given tag depending on which are the neighbouring tags (*context*) on both sides of the word. The computation of the fitness of the individuals is based on the data extracted of a training corpus tagged by hand. These data are organized as contexts. The tagger is able to learn from a training corpus so as to produce a table of rules (contexts) called *training table*. This table records the different contexts of each tag. The table can be computed by going through the training text and recording the different contexts and the number of occurrences of each of them for every tag in the training text.

Results indicate that the evolutionary approach for tagging texts of natural language obtains accuracies comparable to other statistical approaches, while improving the robustness of the typical algorithms used for the same purpose in other stochastic tagging approaches (such as the widely used of Viterbi). These methods typically perform at about the 96% level of correctness [3] (percentage of words correctly tagged). However, there is a limit beyond which no further improvement can be obtained, neither by enlarging the size of the context, nor the size of the training text.

This leads us to investigate the application of messy GAs to the tagging problem with the aim of studying the possible relationships between the tags of the sentence. Statistical methods for tagging rely on the assumption that the only influence to the correctness comes from the words surrounding the considered one. Therefore, in the evolutionary tagger solutions are evaluated according to the position of the genes. In a messy GA, individuals may be composed of any set of genes, therefore relaxing the previous assumption and allowing to investigate other kinds of dependencies among the words to be tagged. Messy GAs do not require all genes to be present in the chromosomes and they may also appear more than one time. This allows us to study the kind of building blocks that arise, thus obtaining information of possible relation between the tag of a word and other tags corresponding to any position in the sentence. Accordingly, this paper presents an extension of the fast messy GAs to a fast messy evolutionary algorithm (EA), which does not work with bit strings but with tag strings for the tagging problem.

1.1 Messy Genetic Algorithms

Messy Genetic Algorithms [5] are variants of Genetic Algorithms that improve the survival of high quality partial solutions or *building blocks*. This is achieved

by explicitly composing increasingly longer and highly fit string from previously obtained and tested building blocks. This is a technique to tackle the problem of building block disruption — *linkage problem* [9] — mainly due to the fixed mapping from the solutions into the representations of individuals or chromosomes and to the way in which the crossover operator combine two chromosomes to obtain a new one. Classic crossover operators often break promising building block leading the algorithm to a local optimum. Furthermore, because messy GAs expedite the presence of high quality building blocks, they increase the probability of exchanging different building blocks [7].

Messy GAs use variable-length strings that are sequences of *messy genes*. A messy gene is an ordered pair composed of a position and a value. The recording of this information allows any building block to achieve tight linkage. Messy GAs do not require all genes to be present in the chromosomes. Therefore, *under* or *overspecified* chromosomes may arise. For example, in the messy string ((1 1) (3 0) (1 0)), the leftmost element specifies gene 1 and its allele value is 1. In a three-bit problem, this chromosome is underspecified, because gene 2 is absent, and overspecified, because gene 1 appears twice. Overspecification is handled by applying a first-come-first-served rule on a left to right order. As for underspecification, some problems do not require to deal with it in any special way because structures of any size can be interpreted and evaluated. Otherwise, the missed genes are filled with those of a *competitive template*, a string that is locally optimal.

A messy GA proceeds in two different phases. The first one, called *primordial* phase, aims to select tight building blocks. It is achieved by initializing the population with all possible building blocks of a specified length. The proportion of good building blocks is improved by applying selection alone for a number of generations. Furthermore, the population size is usually reduced at particular intervals. Thereafter, the next phase, the *juxtaposition* phase, proceeds applying different genetic operators, like *cut* and *splice*, with the objective of recombining the building block obtained in the first phase. In order to deal with chromosomes of variable length, the traditional crossover operator is substituted by the complementary operators cut and splice. Cut divides the chromosome with a specified probability, proportional to the length of the chromosome, while splice joins two chromosomes.

The previous description corresponds to the original messy GA, which suffers from an initialization bottleneck due to the generation of all building blocks of a particular size k: it requires a population of $O(l^k)$ individuals, l being the length of the problem strings. This problem has been dealt with by the socalled *fast* messy GAs [7, 10]. Fast messy GAs use smaller populations composed of longer chromosomes. However these longer chromosomes introduce too much error places [6]. This effect is handled by the mechanism of *building block filtering*. According to this, the process begins with long chromosomes, that are reduced by applying selection and random deletion at specific intervals. Another feature of messy GAs is the use of a *thresholding selection*, which enforces any two

chromosomes to share some threshold number of genes before competing for selection.

The rest of the paper proceeds as follows: section 2 describes the main elements in the fast messy EA for tagging; section 3 is devoted to evaluate the model and section 4 draws the main conclusions of this work.

2 Messy Evolutionary Algorithm for Probabilistic Tagging

The fast messy EA for tagging works with a population of sequences of genes (chromosomes) of variable length. Each gene consists of a position indicating the word of the sentence to be tagged and a tag chosen for that word. A word may appear any number of times in the chromosome or be missing at all. Let us considered the following words and their tags: **Rice**: (NOUN), **flies**: (NOUN, VERB), **like**: (PREP, VERB), **sand**: (NOUN). Then a possible individual when tagging the sentence *Rice flies like sand* is shown in Figure 1. The algorithm uses

Rice	like	sand
NOUN	VERB	NOUN

Fig. 1. Example of an individual for the sentence *Rice flies like sand.*

a *pattern* for the evaluation of the underspecified chromosomes. A pattern is a complete tagging of the sentence. Initially each tag of this pattern is randomly selected with a probability proportional to the frequency of the tag. Then the algorithm performs a number of iterations, each of which is devoted to building up increasingly longer building blocks. The pattern is updated after every iteration step, replacing its tags by those present in the best current individual. Figure 2 shows a pattern for the sentence of the previous example. When the individual of Figure 1 is evaluated, the tag for the word *flies*, which is missing in the chromosome, is taken from the pattern. As in a messy GA, each iteration

Rice	flies	like	sand
NOUN	NOUN	VERB	NOUN

Fig. 2. Example of pattern for the sentence *Rice flies like sand.*

proceeds in three consecutive phases (see Figure 3). In the *initialization* phase, a population of individuals is generated to represent the classes corresponding to each set of genes. In the original messy GA the initial population was composed of a single copy of all substrings of length k. This initialization ensured that all

building blocks of the considered length were present but led to a bottleneck for a problem length l moderately large. This is avoided by a technique called *probabilistically complete initialization* [7], which creates a population of longer individuals ensuring that, with high probability, any building block appears at least once. Two parameters control the procedure: the length of the initial individuals and the population size. Let us call l' the length of the initial individuals, a value larger than k and smaller than l. The results obtained in [7, 8] indicate that if $k < l/2$ and we set $l' = l - k$, a population size $O(l)$ is a good choice. These results have been obtained assuming two alleles for each gene, but they may be extended to an arbitrary number of them. Accordingly we assume these results and take, in our case, the population size as a function of the length of the sentence. Our experimental results serve as an a posteriori justification.

```
function Fast_messy_EA()
    pattern = random_tagging(sentence);
    for level = initial_level to final_level do{
        Probabilistic_initialization(Population, level);
        Primordial_phase(Population, pattern, level);
        Juxtaposition_phase(Population, pattern);
        pattern = update_pattern (pattern, best(Population); }
```

Fig. 3. Scheme of the fast messy EA for tagging

2.1 The Primordial Phase

After the initialization, the *primordial* phase (Figure 4) selects the best individuals of each combination of genes of size k. Because chromosomes in a messy algorithm may contain very different sets of genes, and thus have little in common, a special selection is required to make the competition fair. This is called *thresholding selection*, and it is implemented by applying tournament selection only between two individuals that have in common a number of genes greater than a threshold value. In our case, the number of common genes is computed as the number of different words of the sentence tagged in both chromosomes.

Individual Evaluation The fitness function is related to the total probability of the sequence of tags assigned to the sentence. The raw data to obtain this probability are extracted from the training table. The fitness of an assignment is defined as the sum of the fitness of its positions, $\sum_i (f(g_i))$. The fitness of a position g is defined as

$$f(g) = \log P(T|LC, RC),$$

where $P(T|LC, RC)$ is the probability that the tag of position g is T, given that its context is formed by the sequence of tags LC to the left and the sequence

```
function Primordial_phase(Population, pattern, level)
    level_temp = length(Population[0]) - level);
    while (level_temp > level) do{
        Selection(Population);
        Gene_deletion(Population);
        Evaluation(Population);
        level_temp–; }
```

Fig. 4. Scheme of the primordial phase

RC to the right. This probability is estimated from the training table as

$$P(T|LC, RC) \approx \frac{occ(LC, T, RC)}{\sum_{T' \in \mathcal{T}} occ(LC, T', RC)}$$

where $occ(LC, T, RC)$ is the number of occurrences of the list of tags LC, T, RC in the training table and $\mathcal{T}$ is the set of all possible tags of g_i. The contexts corresponding to the position at the beginning and the end of the sentences lack tags on the left-hand side and on the right-hand side respectively. This event is managed by introducing a special tag, NULL, to complete the context.

In our case chromosomes are evaluated by previously building a tagging of the sentence with tags coming from the individual if it contains any instance of the corresponding gene and from the pattern otherwise. For overspecified genes we take the first occurrence of the gene in a left-to-right order. For example, if we are evaluating the individual of Figure 1 and we are considering contexts composed of one tag on the left and one tag on the right of the position evaluated, the second gene, for which there are two possible tags, NOUN (the one chosen in this individual) and VERB, will be evaluated as:

$$\frac{\#(\text{NOUN NOUN VERB})}{[\#(\text{NOUN NOUN VERB}) + \#(\text{NOUN VERB VERB})]}$$

where # represents the number of occurrences of the context. The remaining genes are evaluated in the same manner.

Another mechanism introduced in this phase is the deletion of genes to reduce the length of the individuals from l' to k. The genes to be erased are randomly selected. We expect to have enough copies of the best building blocks so that after the random deletion some of them still remain.

2.2 The Juxtaposition Phase

The last phase, called *juxtaposition* (Figure 5), is devoted to combine the tight building blocks obtained in the previous phase. This phase also uses thresholding selection as well the *cut-and-splice* operator to combine individuals, and the mutation operator. The rate of application of the cut operator increases with the length of the individuals, while splicing is applied at a fixed rate. Thus, in

```
function Juxtaposition_phase(Population, pattern)
    generation = 0;
    while (generation < max_generation) && not convergence do{
        Selection(Population);
        Cut_splice(Population);
        Mutation(Population);
        Evaluation(Population);
        generation++; }
```

Fig. 5. Scheme of the juxtaposition phase

the beginning of the evolution process, cut-and-splice behaves almost like splice alone, increasing the length of the individuals. Later on, when the length of the individuals is long enough, cut and splice are applied together producing an effect similar to a one-point crossover operator.

Mutation is then applied to every gene of the chromosomes resulting from the cut-and-splice operation with a probability given by the mutation rate. The tag of the mutation point is replaced by another of the valid tags of the corresponding word. The new tag is randomly chosen according to its probability (the frequency it appears with in the corpus).

Chromosomes resulting from the application of genetic operators replace an equal number of individuals selected with a probability inversely proportional to their fitness.

3 Experiments

The algorithm has been implemented on C++ language on a Pentium II PC. Tables 1 and 2 compare the accuracy rate obtained with a classic EA and a fast messy EA for the tagging problem. Experiments have been carried out using a training corpus of 185000 words extracted from the *Brown* corpus [12], a test text of 500 words, population sizes once, twice or three times the length of the sentence to be tagged, a crossover rate of 30%, and a mutation rate of 1%. Table 1 presents the highest accuracy achieved in ten runs for all population sizes; Table 2 presents the average and standard deviation of these ten runs for a population size twice the sentence length, as well as the results of a t-test to asses the statistical significance of the differences in accuracy of both methods.

We can observe that the messy algorithm slightly but systematically improves the results. Although the accuracy results are limited a priori by the statistical information used in the fitness function, the systematic improvement obtained suggests the existence of some correlation between the word to be tagged and distant words, which comes to light by the nature of the messy EA. This hypothesis is supported by the shape of the building blocks obtained from the primordial phase. Figure 6 shows some instances of building blocks obtained from the primordial phase for a sentence of 29 words and a level $k = 7$. A '0' indicates that

	Accuracy						evaluations n.	
	PS = |s|		PS = |s|*2		PS = |s|*3		PS = |s|*2	
	CEA	FMEA	CEA	FMEA	CEA	FMEA	CEA	FMEA
5 generations	95.44	96.40	95.44	96.40	95.20	95.62	98964	11521
10 generations	95.44	96.48	96.16	96.88	95.68	96.64	102944	21494
20 generations	95.68	96.88	96.16	97.12	96.16	96.68	109964	38136
30 generations	95.44	96.88	95.92	96.88	95.92	96.64	117337	51225
40 generations	95.68	96.40	95.92	96.88	95.92	96.64	123757	60728

Table 1. Largest accuracy of ten runs obtained with the classic EA and with the fast messy EA with different numbers of generation and population sizes for the tagging problem. CEA stands for classic evolutionary algorithm and FMEA for fast messy evolutionary algorithm. In the FMEA the number of generations refers to the juxtaposition phase. The range of sizes of building blocks in the primordial phase has been |s| / 3.5 – |s| / 2.5. The threshold value to allow competition during selection has been |s| - 2, and the threshold value to begin to apply *cut* has been |s| / 1.5. Two last columns show the number of fitness evaluations for PS = |s|*2.

	Building blocks	Fitness
1	022001210000010220000302	20.2156
2	002000202202000200200302	22.1340
3	002002200020201200000202O	21.6027
4	000020002000222002020230O	21.6340
5	002321000002001020202000O	20.7156
6	022120000000221002000202 2	21.1027
7	020320012002000200002000 2	21.1352
8	002021200022000020002002O	21.2468
9	002000222000220000002030 2	22.1340

Fig. 6. Samples of building blocks obtained from the primordial phase for a sentence of 29 words and a level $k = 7$. Average fitness is 20.92

the gene corresponding to the word of that position in the sentence is absent and any other number that it is present at least once. A '1' indicates that the gene is present but the tag is wrong, '2' that the tag is right and '3' that the tag is right while it was wrong in the tagging performed with a classic EA. We can observe that the worse building blocks present very sparse genes. We can also observe that although some of the best building blocks, such as 9, present their genes very grouped, there are also some others, such as 2, with some sparse genes. Furthermore, we can also observe that some of the tags that have been correctly assigned by the messy EA but wrongly by the classic EA (marked '3') appear among sparse genes. The evaluation of contexts containing missing genes amounts to taking the tag of the pattern and thus, in general, at least in the primordial phase, the resulting context will be composed of tags without any relation. For the same reason, we expect that consecutive genes increase the fitness

	CEA		FMEA		significance
	Mean	St. dev.	Mean	St. Dev.	t-test signif.
5 generations	95.5	0.441	96.1	0.415	0.0057
10 generations	95.5	0.411	95.8	0.343	0.17
20 generations	95.5	0.435	96.3	0.509	0.0015
30 generations	95.4	0.518	96.1	0.610	0.020
40 generations	95.5	0.499	96.5	0.495	0.0002

Table 2. Study of the statistical significance of the results obtained for a population size of $2^*|s|$. CEA stands for classic evolutionary algorithm and FMEA for fast messy evolutionary algorithm. Last column reports the probability that the two samples (the accuracies of the ten runs with the CEA and those of the FMEA) arise from the same probability distribution (a small number means that the difference in the means is statistically significant).

because with high probability they correspond to frequent contexts (they have appeared after the selection of the primordial phase). Therefore, the selection of chromosomes with separate genes may indicate a correlation between tags by some hidden indirect mechanisms. To illustrate this assume the following pattern for a sentence with n words:

$$P : P_1, P_2, \cdots, P_n$$

where P_i is the tag assigned to the word w_i. Let us consider now the following individuals (__ denotes a missing gene), and let T represent the tag they have assigned to the word:

$$I1 : T_1, T_2, T_3 \cdots, \_$$
$$I2 : T'_1, \_, T'_3, \_\_, T'_5, \cdots, \_$$

Assuming that a context is composed of one tag on the left and one of the right, the evaluation of $I1$ is done according to the frequencies of the contexts (T_1, T_2), (T_1, T_2, T_3), (T_2, T_3, T_4), etc, that must have high frequencies because they have been selected. However, the evaluation of $I2$ is done with the contexts (T'_1, P_2), (T'_1, P_2, T'_3), (P_2, T'_3, P_4), etc, composed of tags not necessarily related. Therefore, in general, they will have low frequencies and will hardly appear. Accordingly, the actual occurrence of an individual as the following one

$$I : T''_1, \_\_, \_, T''_4, T''_5, \cdots, \_$$

may indicate a hidden relation between T''_1 and T''_4, T''_5 (e.g. they might correspond to words separated by a subordinate phrase, or by words that have only one possible tag). In this way the messy EA although less efficient than classic EAs (the number of fitness evaluations is more than twice, see two last columns of Table 1), helps to uncover hidden relations between the words of the sentence.

3.1 Study of the Algorithm Parameters

Two fundamental parameters for the probabilistic initialization phase in a messy EA are the length of the chromosomes and the size of the initial population. Some experiments have been carried out in order to determine the most appropriate values of these parameters in our problem. Because the text is tagged sentence by sentence it seems reasonable to adjust these parameters as some kind of simple function of the length of the sentence. Figure 7 shows the results obtained when varying the length of the chromosomes for different population sizes. All parameters are established as a function of the length of the sentence. We can observe that values of the order of the problem length are enough. A parameter

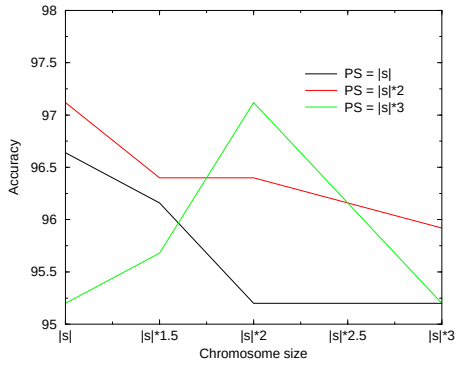

Fig. 7. Accuracy obtained with different sizes of the chromosomes. Each chart corresponds to a different population size: P1 stand for a population size of equal to the length of the sentence, P2 to one of twice the length of the sentence and P2 of three times this length.

of the algorithm that needs to be studied is the threshold value for the measurement of similarities between two chromosomes, in order to decide if they are similar enough for the comparison to make sense. Table 3(a) shows the results for some values defined as a function of the length of the sentence to be tagged. The number of similarities is measured as the number of different words of the sentence that corresponds to any of the genes of the chromosome. The best result is obtained when the required similarity is equal to the length of the sentence minus 2. This indicates that only very similar chromosomes must be compared.

Another parameter to be fixed in a messy EA is the threshold value for the chromosome size to begin to apply the *cut* operator. Table 3(b) shows the results. The best accuracy is obtained when cut is only applied to individuals longer than two thirds of the length of the sentence.

Finally, the range of sizes of building blocks explored in the primordial phase has also been studied. Table 3(c) shows the results obtained. In this case, the

Thres. similarity	Acc.
|s|	96.16
|s| - 1	96.64
|s| - 2	97.12
|s| - 3	95.44

(a)

Thres. length	Acc.
|s| / 1.3	95.68
|s| / 1.4	96.88
|s| / 1.5	97.12
|s| / 1.6	96.16
|s| / 1.7	95.68

(b)

Range	Acc.
|s| / 3 – |s| / 2	96.40
|s| / 4 – |s| / 3	96.88
|s| / 3.5 – |s| / 2.5	97.12
|s| / 3 – |s| / 2.5	96.88

(c)

Table 3. Table (a) presents the accuracy obtained with different values of the threshold established to considerer two chromosomes similar enough to be compared. Table (b) shows the accuracy obtained with different values of the threshold size of the chromosomes to apply the *cut* operator. Table (c) presents the accuracy obtained with different range of sizes of building blocks in the primordial phase. |s| stands for length of the sentence. When varying one of the three parameters the other two are assigned the value marked in boldface.

greater accuracy is obtained when the size of the building blocks explored ranges between the length of the sentence divided by 2.5 and the length divided by 3.5.

4 Conclusions

This work has investigated the kind of dependencies between words in the tagging problem, that is the assignment of lexical categories to the words of a text. This have been done by applying a fast messy evolutionary algorithm to solve the problem. This algorithm is an extension of the fast messy genetic algorithm, a variety of Genetic Algorithm that improves the survival of high quality partial solutions or *building blocks*. Thus, it is a technique to tackle the problem of building block disruption or *linkage problem* partially due to the fixed mapping from the solutions into the representations of individuals. In a messy GA individuals are variable-length strings of *messy genes*. A messy gene is an ordered pair composed of a position and a value. The recording of this information allows any building block to achieve tight linkage. These algorithms allow obtaining in a phase, called *primordial phase*, previous to the evolutionary process, a set of tight building blocks, whose features can provide information about the internal dependencies of the problem.

Results obtained for the tagging problem systematically outperform a little those obtained with a classic evolutionary algorithm, thus indicating the existence of other relation between tags apart from those between the neighbouring words. This idea is also supported by the kind of building blocks obtained from the primordial phase. Some of them present their genes grouped, but in others the genes are mainly sparse in the proximity of the group or forming others groups.

These results indicate the presence of more complex relationships between words. Accordingly, in the future those possible relationships will be investigated by studying the dependencies between the tagging and the parsing problem.

A number of parameters affecting the performance of the results have also been investigated. These parameters have been assigned values as a function of the length of the sentence. The study of the size of the chromosomes and of the initial population size in the probabilistic initialization phase indicates that values close to the length of the sentence or twice this length are enough to obtain the best results. The experiments on the threshold value to consider that two chromosomes are comparable indicates that very similar chromosomes are required (different at most by the presence of two genes).

References

1. L. Araujo. A parallel evolutionary algorithm for stochastic natural language parsing. In *Proc. of the Int. Conf. Parallel Problem Solving from Nature (PPSNVII)*, 2002.
2. E. Brill. Transformation-based error-driven learning and natural language processing: A case study in part of speech tagging. *Computational Linguistics*, 21(4), 1995.
3. E. Charniak. *Statistical Language Learning*. MIT press, 1993.
4. D. Cutting, J. Kupiec, J. Pedersen, and P. Sibun. A practical part-of-speech tagger. In *Proc. of the Third Conf. on Applied Natural Language Processing*. Association for Computational Linguistics, 1992.
5. D.E. Goldberg, Korb B., and Deb K. Messy genetic algorithms: motivation, analysis, and first results. *Complex Systems*, 3:493–530, 1989.
6. D.E. Goldberg, Korb B., and Deb K. Messy genetic algorithms revisited: Studies in mixed size and scale. *Complex Systems*, 4:415–444, 1990.
7. D.E. Goldberg, Kargupta H. Deb K., and Harik G. Rapid, accurate optimization of difficult problems using fast messy genetic algorithms. In *Proc. of the Fifth International Conference on Genetic Algorithms*, pages 56–64. Morgan Kaufmann Publishers, 1993.
8. D.E. Goldberg, Deb K., and J. H. Clark. Don't worry, be messy. In *Proc. of the Fourth International Conference in Genetic Algorithms and their Applications*, pages 24–30, 1991.
9. Georges R. Harik and David E. Goldberg. Learning linkage. In Richard K. Belew and Michael D. Vose, editors, *Foundations of Genetic Algorithms 4*, pages 247–262. Morgan Kaufmann, San Francisco, CA, 1997.
10. H. Kargupta. *Search, polynomial complexity, and the fast messy genetic algorithm*. Ph.D. thesis, Graduate College of the University of Illinois at Urbana-Champaign, 1996.
11. B. Merialdo. Tagging english text with a probabilistic model. *Computational Linguistics*, 20(2):155–172, 1994.
12. Francis W. Nelson and Henry Kucera. Manual of information to accompany a standard corpus of present-day edited american english, for use with digital computers. Technical report, Department of Linguistics, Brown University., 1979.
13. H. Schutze and Y. Singer. Part od speech tagging using a variable memory markov model. In *Proc. of the 1994 of the Association for Computational Linguistics*. Association for Computational Linguistics, 1994.