Evolutionary Sentence Building for Chatterbots

Dana Vrajitoru
IUSB, Computer and Information Sciences
1700 Mishawaka Ave,
South Bend, IN 46634, USA
email:danav@cs.iusb.edu

Abstract

Chatterbots are computer programs that simulate intelligent conversation. They make use of various techniques such as pattern matching, indexing, sentence reconstruction, and even natural language processing. In this paper we present an approach to chatterbots that mixes pattern matching with indexing and query matching methods inspired from information retrieval. We propose a model in which new sentences can be produced from existing ones using an evolutionary algorithm adapted to the structure of the natural language.

1 INTRODUCTION

Chatterbots are computer programs that simulate intelligent conversation. The typical execution of the program involves an input from the user in natural language to which the program provides an answer that should sound like a reasonable and possibly intelligent response to the original sentence. The whole process is repeated while the human keeps the conversation going.

The very first chatterbot, named Eliza [18], was written by J. Weizenbaum in 1995, and simulated a Rogerian psychotherapist. The idea was very simple and consisted in a pattern matching algorithm and sentence reconstruction following templates, with no in-depth knowledge or processing of the natural language. The program proved to be amazingly efficient in sustaining people's attention during the conversation and the success of the original program has influenced the development of many others.

Using similar ideas, Colby from Stanford AI Lab developed *Parry*, the paranoid, in 1971. Parry is the opposite of Eliza and has been intended as a study of the nature of paranoia and is capable of expressing beliefs, fears, and anxieties [1], [6].

Among the famous chatterbots we can mention Racter [4] by W. Chamberlain and T. Etter, who has been attributed the authorship of the book *The Policeman's Beard Was Half Constructed*, and is basically a story teller. The authenticity of the book has been questioned afterward [2].

Another chatterbot worth mentioning is A.L.I.C.E, Artificial Linguistic Internet Computer Entity [9], that has its own development language called AIML (artificial intelligence markup language) and earned the Loebner Prize, based on the Turing Test [15], in 2000 and 2001.

The most recent development are the so-called virtual agents that can enhance the content of the web page and help guiding the customer [10]. Such agents incorporate knowledge about the company owning them and are competent in answering business related questions [10].

The goal of the chatterbots we have implemented is to simulate particular personalities, either fictional or real, mostly taken from literature, film, or television shows. We start from the idea that we dispose of a database of sentences that can be attributed to the personality to be simulated, as for example, the text of the book or their lines from a script. The first prototypes can be found online [16].

In our basic model, we build the answer to the human's input by a probabilistic choice between pattern matching and sentence building based on templates, which is similar to Eliza, and query matching in a set of sentences indexed by keywords, inspired by information retrieval. Our model also keeps track of the short-term history and makes sure that the same answer is not given twice in a row, even if the human keeps repeating their input.

In this paper we also introduce an additional component of the system, in which we employ an evolutionary approach to enhance the diversity of the chatterbot's answers. In this model, we start by retrieving a whole set of sentences as potential answers the human's input and we combine them to produce new sentences using a method similar to genetic algorithms. The fitness function is based on the similarity between the generated sentences and the original human sentence. One sentence is chosen in the end by a probabilistic function and is presented to the user. This way we hope to generate new sentences that follow the conversation style of the original character, but have not yet been attributed to it.

The paper is structured the following way. Section 2 gives some details about the general techniques that we have implemented for our chatterbots. Section 3 introduces the general algorithm for a virtual character based on these techniques. Section 4 introduces the evolutionary algorithm used to generate new sentences from the ones already existing in the database. The paper ends with some conclusions.

2 CHATTERBOT TECHNIQUES

In this section we briefly introduce the main techniques used by chatterbots, which are pattern-matching and indexing.

2.1 PATTERN MATCHING AND TEMPLATES

The pattern matching technique consists in finding one or several patterns that match the sentence entered by the the user. A pattern is generally defined as a sentence in natural language in which certain parts have been replaced by wild cards that can be matched to any group of words in a matching sentence.

For each pattern that is defined in the database, a corresponding template is utilized to generate the answer to the sentence. The parts from the original sentence that are identified with the wild cards are first subjected to a person transformation in which words like "I, my, mine" become "you, your, yours" and the other way around.

For example, a pattern in the original Eliza program can be expressed as

*I want **

in which the '*' character can be replaced by any sequence of words. The corresponding template to generate the answer can be expressed by

*What would it mean to you if you got **

in which the '*' is replaced by the sequence of words that has been matched to the wild card in the pattern.

An application of this pattern could be the following dialog:

User: I want to know how it works.

Eliza: What would it mean to you if you got to know how it works?

The next dialog based on the same pattern is an example of the necessity of the person transformation:

User: I want to talk to my adviser.

Eliza: What would it mean to you if you got to talk to your adviser?

Beside the list of patterns, the original Eliza program also contained a list of sentences that can be given in answer to sentences that cannot be matched to any of the patterns, like:

What does that suggest to you ?

Please go on.

For a virtual character, the patterns are built from the database of character lines from the book or the script. They are based on the lines of any other character that precedes the character we are developing in the dialog. and the response templates are generated from the character's answer in the original dialog. For the moment, the patterns have been chosen manually, and as future work we intend to implement an automatic pattern and template generator.

Following the ideas in the original Eliza, our chatterbots also contain a list of random sentences that can be answered to an affirmative-type or a question-type input from the user.

2.2 INFORMATION RETRIEVAL AND CHATTERBOTS

To begin the discussion about applying Information Retrieval (IR) techniques to chatterbots, we can remark that there are some similarities between these two applications, even though their goals are different.

In the classical IR approach [11], we are given a collection of documents (ASCII text in natural language) and a query expressed by a human in natural language. The task of the system is to find the documents in the collection that are the closest match to the given query.

We can extend this problem to the chatterbot or virtual character applications by considering that each document consists of one or two sentences associated with the character in the original script. In this approach, we can regard the sentence entered by the human participant in the dialog as being the query. This input sentence can be expressed in either affirmative, negative, or interrogative form. In our case we want to find one particular document (sentence) that can be seen as a good answer to the query.

Although the initial ideas are similar, there are also some differences between the two approaches implying that we must modify the IR automatic indexing and retrieval techniques before applying them to chatterbots.

First, in IR the goal is to provide the user with a reasonable number of documents presented in an order corresponding to their relevance to the query. In the virtual character model, the goal is to provide the user with just one, eventually two of the documents - sentences in the collection that can be interpreted by the user as a coherent answer to his input.

Second, while in IR we hope that all of the terms in the query will be present in the retrieved documents, the definition of a coherent answer for chatterbots is much more general and subjective. The response may be satisfactory from the point of view of the human participant even if it doesn't contain any of the terms from the input sentence (query). We can assume, however, that the presence of some of these terms in the returned phrase would increase the human's perception of it as an intelligent answer.

The common ground between the two models suggests that we can use some of the methods developed in IR to improve our system, like the automatic indexing, query matching and classification of the retrieved documents according to their relevance to the query. These techniques are especially useful when we dispose of a large number of sentences that can be associated with the character we intend to simulate. For example, if we start from a given character taken from a book, movie, or television show, that initial collection of sentences would be the transcript of all the parts in the book or script that constitute the character's lines or contribution to the dialog.

The first IR method that we can apply to chatterbots is the indexing of documents - sentences by keywords that represent significant words figuring in the text of the documents. In the current implementation of our chatterbots, we have used a manual indexing, and an automatic one is under development.

The automatic indexing starts by removing all of the common words from each document, such as "a", "is", "for". This part must be adapted to the chatterbot approach in the sense that some words that are considered insignificant in IR, can be very meaningful in a dialog, like "you", "because", etc. The automatic indexing system we are developing utilizes two lists of stop words, some of them that define the semantics of the sentence and thus must be taken into consideration in a particular way, and others that can simply be ignored.

The second step in automatic indexing is to remove all the unnecessary suffixes, like the plural "s", the "ing" from the continuous form of verbs, etc. This step insures that words from the same family, like "drive", "driving", "driver", will be indexed as a single keyword.

In the third step, the system creates an index of the collection of documents based on the resulting terms. In general, each term is assigned a weight based on its frequency in the document as well as in the entire collection. This is known as the vector space model [11]. Given the reduced length of our documents, a Boolean indexing [12] is appropriate. In other words, we record only the presence or absence of a term in an indexed sentence, with no assumption as to its importance.

In our model, the documents (sentences) are indexed not only by terms occurring in them, but also by some of their synonyms. Sometimes words that appear in the text before the sentence being processed can also be significant to it, so our indexing also takes into account the context of a document.

The second technique from IR that we can apply to the virtual character is the automatic document-query matching and document retrieval. For this, we can consider the sentence entered by the user as the query.

In the classical IR approach, the query is processed the same way as the documents, meaning that all of the common words are removed as well as the suffixes, and the remaining terms are indexed based on their frequency. In our model, we also have to take into account the two stops lists for this phase.

In the retrieval process, the system selects all of the documents (sentences) that contain at least one of the terms in the query (the user's input) and rank them according to the number of terms in common with the query. This is a special case of the cosine formula for document-query matching introduced by Salton [11], applied to the case where the indexing is Boolean. In IR, the system will return all of the documents matching the query in the order of their similarity to the query. In our case, we select only one of the matching sentences based on a probabilistic function.

This approach increases the efficiency of the chatterbot and organizes the knowledge included in the program in a way that can also be extended to other applications. The limitation of the system is that it cannot generate sentences that did not exist in the original database and this reduces the potential diversity of the answers.

The collection of sentences is given in the beginning and each of them is recorded in its initial form. If the collection is big enough, this may be sufficient for a one-time conversation, but a recurrent user of the system may find it repetitive. More sentences can be added to the collection at a later point, if this seems necessary, but once a sentence has been indexed, it cannot change. Thus, a person knowing very well the initial character can get the impression that the virtual character is only repeating word by word, sentences that were already present in the original script.

A combination between this technique and the pattern-matching with template answers is a partial solution to the problem, and this is the actual state of our online chatterbots [16]. The approach that we introduce in this paper is to add a third technique that can dynamically generate new sentences and eventually add them to the database, and this is the evolutionary computation model that we present in Section 4.

3 THE VIRTUAL CHARACTER

The chatterbot algorithm can generally be described by the following pseudocode:

```
while (true) {
  Read(input);
  output = Make_answer(input);
  Write(output);
}
```

The Make_answer function processes the sentence from the user and generates an answer based on the following rules:

- It starts by looking for a pattern with a given probability (for example, 90%).
- If several patterns match the input, it selects one of them randomly and generates the answer based on the template associated to it.
- If the previous step has been unsuccessful, it looks for a sentence indexed by any of the terms present in the input sentence. Another probability is associated with this second step (90% again).
- If the answer could not be generated from the previous steps, it selects it from a list of random answers. The case where the input sentence was formulated as a question is treated separately in this step.

3.1 SHORT-TERM MEMORY

As a requirement of a seemingly intelligent conversation, a chatterbot should include some implementation of a short-term memory that prevents it from repeating what it is saying, even if the human keeps feeding it the same input sentence.

In our current model, the chatterbot keeps track of the latest sentence that has been generated and keeps calling the Make_answer function until the new sentence is different from the last one. We intend to extend this memory to include the latest pattern that has been used instead of simply the resulting sentence and also to go a few steps back.

The short-term memory of a chatterbot can also serve to create a certain coherence in the conversation. As a future work, we intend to keep track of a general context of the conversation containing a given number of terms that have been used in the dialog and include them in the search for a new answer. This way, the chatterbot would not only be conversing on the current topic, but also on topics that have been mentioned a few sentences back.

3.2 FEEDBACK

To evaluate a virtual character we must verify if several constraints are fulfilled. Thus, the user can be asked to provide feedback on the answer to their input sentence and decide whether the answer is

- a syntactically and semantically correct sentence,
- a sentence that is consistent with the simulated character,
- a good or interesting reply to the input sentence.

Some initial experiments with our online chatterbots [16] have shown that about 93% of the answers were syntactically correct, 77% were in character, and 57% could be considered good answers.

4 EVOLUTIONARY ALGORITHM FOR SENTENCE GENERATION

In this section we introduce an evolutionary algorithm that allows us to generate new sentences based on the ones retrieved from the database and enhance the diversity of response of the chatterbot.

Evolutionary computation is not a very common method for natural language processing. Some application of the genetic algorithms to this field include adaptive word segmentation [8], grammar learning [3], linguistic classification [13], and information retrieval [5], [17].

4.1 OUR MODEL

The combination of the previous methods may seem enough to generate interesting and diverse conversations with a human opponent. We would still like the system to be able to learn new sentences during the conversation and possibly grow the database according to some feedback from the user.

Our idea is to use a form of genetic algorithms (GAs) [7] to build new sentences based on the ones that were already present in the database.

The GAs start with a usually random initial population of potential solutions to the problem (individuals) and attempt to build better ones by recombination and mutation operators. Each individual is evaluated to estimate its fitness to the problem being solved. The chances that a particular individual is chosen for reproduction depend on this evaluation.

In our case, the initial population consists of the sentences retrieved by the system as potential answers to the input sentence from the user. The fitness function is computed as the number of terms that the sentence has in common with the query (input). We have used the crossover operator to generate new individuals exclusively, leaving out the mutation operation.

The difficult part in the algorithm is representing a sentence expressed in natural language as a genetic individual, defining the genes and the building blocks, and choosing the crossover form. The challenge is to recombine two sentences and create new ones that are still making sense with a relatively simple algorithm not requiring a complex natural language parser.

Our idea was to use what we call the *largest common pattern* to convert two sentences into a genetic representation such that we can apply a crossover operator to them.

Definition. Let s and t be two sentences in natural language. The largest common pattern of s and t, denoted by $LCP(s,t)$ is the longest sequence of words or groups of words

$$LCP(s,t) = (p_1,\ p_2,\ \ldots,\ p_n)$$

such that either $n = 1$ and $p_1 = \emptyset$, or $\forall i,\ 1 \le i \le n,\ p_i \ne \emptyset$, and that the two sentences can be expressed as

$$\begin{aligned} s &= s_0\ p_1\ s_1\ p_2\ s_2\ \ldots\ s_{n-1}\ p_n\ s_n \\ t &= t_0\ p_1\ t_1\ p_2\ t_2\ \ldots\ t_{n-1}\ p_n\ t_n \end{aligned}$$

In this definition, we consider a word to be any string composed of alphabetic characters that is delimited by spaces or punctuation characters such as commas, semicolons, and so on. We are looking for the longest pattern in terms of the largest number of characters in the union of $p_1,\ \ldots,\ p_n$, and not of the largest possible n. The substrings s_i and t_i, $0 \le i \le n$, can eventually be empty.

To determine the largest common pattern between two sequences of words we can use an algorithm that is similar to the method determining the distance between two strings, in which we replace each character by a word. Let $s_\star$ and $t_\star$ be the very first two words in the sentences s and t. Then we can define $LCP(s,\ t)$ recursively in the following way:

$$LCP(s,\ t) = \begin{cases} s_\star \cup LCP(s\backslash s_\star,\ t\backslash t_\star), & \text{if } s_\star = t_\star, \\ \max \begin{cases} LCP(s\backslash s_\star,\ t), \\ LCP(s,\ t\backslash t_\star), \\ LCP(s\backslash s_\star,\ t\backslash t_\star), \end{cases} & \text{if } s_\star \ne t_\star, \\ \emptyset, & \text{if } s = \emptyset, \\ \emptyset, & \text{if } t = \emptyset \end{cases}$$

In this formula, we have denoted by $s\backslash s_\star$ and $t\backslash t_\star$, the sentences that we obtain by removing the first word from s and t respectively.

The meaning of the first line in the formula is that the LCP between two sentences starting with the same word contains that word plus the largest common pattern between the remaining of both sentences. The second line means that if the first two words are different, then the LCP can be computed as the maximum between the largest common pattern obtained by ignoring the first word either in the first sentence, or in the second sentence, or in both. The last two lines mean that the recursive algorithm stops when any of the two sentences is empty.

As soon as we have determined the LCP of the two sentences, we can define their genetic representation to be the complement vector of the LCP in each sentence. Thus, if we denote by Ind_s and Ind_t the individuals representing each of these sentences, we can define them by the following:

$$Ind_s = (s_0,\ s_1,\ s_2\ \ldots,\ s_{n-1},\ s_n)$$
$$Ind_t = (t_0,\ t_1,\ t_2,\ \ldots,\ t_{n-1},\ t_n)$$

The genes in these individuals represent the words or groups of words that are different in the sentences and that are situated between each two common words or groups of words composing the *LCP*. Note that the genetic representation of each sentence depends on the second sentence that is compared to it and changes from one crossover operation to the next.

Since in this model the two sentences have the same number of genes, we can apply any of the classical crossover operators. We have chosen the uniform crossover [14]. This operator exchanges the corresponding genes in the two individuals for each position independently with a swap probability of 0.5 in our case. In this particular case, each pair of genes $(s_i,\ t_i)$, where $0 \leq i \leq n$, can be independently swapped.

After the swapping step, the child sentences are generated by reinserting the words or group of words from the LCP in the resulting individuals in the same places they were in the initial sentences.

In our model, the crossover can only be applied to sentences with the same final punctuation mark. Thus, a question can only be combined with another question and the same rule applies to affirmative sentences.

In this model, the genetic representation of the sentences has to be created and disassembled for each crossover operation, which is more time consuming than for the classical GAs where the representation of a particular solution to the problem is unique. On the other hand, this application doesn't require generating a large number of new individuals. The entire operation of genetic evolution is also limited by the time the user is willing to wait fort the system to reply.

The entire process of generating new sentences for a particular query follows an evolutionary algorithm in which the new sentences are added to the existing ones and the process continues up to a given population size. We have chosen this model because of the limited number of individuals available to begin with, and because the coherence of the sentences may decreases with the number of generations, thus the original individuals can still be useful.

Once a given number of new individuals have been generated, one of them is chosen by a random process favoring the sentences of high fitness, and is presented to the user as the answer. This phase is still in experimentation and hasn't yet been included in our online chatterbots.

4.2 EXAMPLE

For a better understanding of our algorithm, suppose that we have the following sentences that we want to recombine using the LCP-based crossover.

$s =$ *You don't have all the information.*

$t =$ *We have a situation that requires all of the resources.*

The first step is to compute the LCP between the two sentences:

$$LCP(s,t) = (have,\ all,\ the)$$

Thus, the two individuals are

$$Ind_s = (You\ don't,\ \emptyset,\ \emptyset,\ information)$$
$$Ind_t = (We,\ a\ situation\ that\ requires,\ of,\ resources)$$

Each of these individuals contains 4 genes. The uniform crossover operator can choose to swap each of them independently with a probability of 50%. Supposing that the decision is made to swap the first and the fourth gene, we obtain the following individuals:

$$child_1 = (We,\ \emptyset,\ \emptyset,\ resources)$$
$$child_2 = (You\ don't,\ a\ situation\ that\ requires,\ of,\ information)$$

By merging these individuals with the LCP of the original sentences, we obtain the following new sentences

We have all the resources.

You don't have a situation that requires all of the information.

We can see that both of these sentences are correct from a syntactic point of view, and they are similar enough to the original ones that they can still be attributed to the same author or character.

Not any recombination of sentences turns out into coherent English phrases. For example, a possible recombination of the following sentences:

Is this what you walked in here for?

What makes you think that?

would result in the following questions that are not syntactically correct:

Is this what you think that?

What makes you walked in here for?

As part of our future work, we would also like to define some general rules for deciding which sentences can be combined based on their structure. For example, a question should only be recombined with another question, but the restrictions could be deeper than that.

For the moment, about 50% of the generated sentences can be considered syntactically and semantically correct.

5 CONCLUSIONS

In this paper we have presented an evolutionary algorithm that can be utilized to enhance the diversity of response of a chatterbot or virtual character. The algorithm is still in experimental phase, but an early prototype of the chatterbot program is already functional online.

The genetic recombination is capable of generating new sentences and enhancing the diversity of the chatterbot while keeping the general style of the original character being simulated. As a down side, the recombination doesn't always result in coherent sentences and further constraints are necessary to improve the success rate of the system.

References

[1] *The Thinking Computer.* Freeman, New York, 1976.

[2] J. Barger. "the policeman's beard" was largely prefab! *The Journal of Computer Game Design*, 6, 1993.

[3] A. Belz. PCFG learning by nonterminal partition search. In H. Fernau and M. van Zaanen, editors, *Grammatical Inference: Algorithms and Applications. Proceedings of the 6th International Colloquium on Grammatical Inference (ICGI 2002)*, pages 14–27. Springer, 2002.

[4] W. Chamberlain. *The Policeman's Beard is Half Constructed.* Warner Books, 1984.

[5] H. Chen. Machine learning for information retrieval: Neural networks, symbolic learning, and genetic algorithms. *Journal of the American Society for Information Science*, 46(3):194–216, 1995.

[6] G. Güzeldere and S. Franchi. Dialogues with colorful personalities of early AI. *Stanford Humanities Review*, 4(2), 1995.

[7] J. H. Holland. *Adaptation in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, 1975.

[8] D. Kazakov and S. Manandhar. Unsupervised learning of word segmentation rules with genetic algorithms and inductive logic programming. *Machine Learning*, 43:121–162, 2001.

[9] The A.L.I.C.E Artificial Intelligence Foundation. http://www.alicebot.org/.

[10] eGain Communications Corp. Web self service. http://www.egain.com/.

[11] G. Salton, editor. *The SMART Retrieval System - Experiments in Automatic Document Processing.* Prentice-Hall, Englewood Cliffs (NJ), 1971.

[12] G. Salton, E. Fox, and U. Wu. Extended Boolean information retrieval. *Communications of the ACM*, 26(12):1022–1036, 1983.

[13] E. V. Siegel and K. R. McKeown. Learning methods to combine linguistic indicators: Improving aspectual classification and revealing linguistic insights. *Computational Linguistics*, 26(4):595–627, 2000.

[14] G. Syswerda. Uniform crossover in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the International Conference on Genetic Algorithms*, San Mateo (CA), 1989. Morgan Kaufmann Publishers.

[15] A.M. Turing. Computing machinery and intelligence. *Mind*, 59(236):433–460, 1950.

[16] D. Vrajitoru. Chatterbots web page. http://www.cs.iusb.edu/˜danav/chatterbots/.

[17] D. Vrajitoru. Crossover improvement for the genetic algorithm in information retrieval. *Information Processing and Management*, 34(4):405–415, 1998.

[18] J. Weizenbaum. Eliza - a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, (9), 1966.