

Inducing a Context Free Grammar for a Natural Language with a Genetic Algorithm

Joseph Lewis

Benjamin Collar

December 14, 2000

Introduction

In this paper we describe the components and operation of a program that induces the grammatical structure of a sample of natural language.

Description of the problem

For a piece of well-structured data, it is possible to program a computer to induce, or discover, a representation of the structure. For language, this structure is often represented as a context free grammar. The program is trained on some samples of the text, then evaluated. General techniques for syntactic induction are given in [3]. These techniques heavily rely on the presence of a teacher who has detailed knowledge of the language.

The problem becomes very difficult when the data are samples of natural language. English, for instance, presents some grammatical structures, like relative clause attachment, that are highly ambiguous. Ambiguity in this instance means that one's understanding of the syntactic structure of a sentence is informed by linguistic data beyond syntax. Some examples of the problems that must be dealt with can be found in [Barbara's paper].

Overview of the program

We propose that a genetic algorithm, with some assistance, can solve this problem. The program will be trained on a set of simple examples. Over time, the samples will increase in complexity. While structural elements that have been previously seen persist in both the internal representation and the sample, new structure is introduced, so the program must continue to adapt the representation. New structure will be introduced slowly, so the algorithm has sufficient time to adjust.

The algorithm will be informed by a constituent guessing component. This component, over time, gathers statistical data regarding known syntactic structure. When the algorithm is faced with a sentence that has unknown syntax, it must guess. This component makes the guess a bit more educated than random choice.

Previous work

Similar experiments have been performed, but have limited scope. They prove the ability of genetic algorithms to induce the structure of many kinds of well-formed data, but only describe small experiments with natural language. Descriptions can be found in [6] and [8]. The authors of these pieces note that NL induction is very difficult.

Experiments have been performed that infer rules for discrete finite automata, which solve the same problem as context free grammars, but the authors have not yet explored the literature in this area.

Context Free Grammars

Context free grammars are a mathematical way of modeling constituent structure [4]. Constituents are groupings of words in a sentence. Example constituents in English are noun phrase and relative clause. A grammar is a 4-tuple consisting of a set of non-terminals, a set of terminals, a set of productions, and a start symbol. The terminals are the words in the language. Non-terminals are labels given to constituents.

A production consists of a left hand side, which is a non-terminal, and a right hand side. The right hand side is a list of one or more terminals or non-terminals. Productions are rules that describe how to translate a sentence of terminals into a tree of non-terminals and terminals that expresses the sentence's structure; this is called a parse tree.

In context free grammars, the left hand side is guaranteed to have exactly one symbol. There are other kinds of grammar with different constraints. Context sensitive grammars allow more than one symbol on the left hand side; this means that whether and how one applies productions depends on the surrounding symbols. Productions are often written as the left hand side, then an arrow, then the list of right hand side. For example:

$$\begin{aligned} S &\rightarrow A \\ A &\rightarrow AB \\ B &\rightarrow 0A \\ B &\rightarrow 1 \end{aligned}$$

A grammar can both produce and parse sentences. A productive grammar begins at the start symbol, S in this case. It then replaces all the symbols on its right hand side with their right hand sides and continues until all symbols have been transformed into terminals. In our example, the terminals are 1 and 0. If there is more than one rule for a given symbol, a rule is chosen randomly.

Parsing is the opposite operation. Given a sentence, the parser builds a tree based on the rules of the grammar that describes the syntax of the sentence. If no tree can be made, the sentence is declared ungrammatical. Parsers and grammars are described in more detail in [4].

A facet of context free grammars that concern us is grammar equivalence. For any context free language there are multiple grammars. There are two kinds of equivalence: weak and strong. Two grammars are strongly equivalent if they generate the same set of strings and assign the same phrase structure to each sentence. Two grammars are weakly equivalent if they simply generate the same set of strings. In this experiment, the grammars represented by individuals will be weakly equivalent. Strong equivalence is not required for successful parsing. The grammar that a machine induces will likely be significantly different than a grammar we, as humans, would derive. This does not imply that the machine's grammar is wrong; it is simply more suitable to the machine's understanding of the syntactic qualities of the text. Our solution to the grammar equivalence problem is presented below.

Genetic Algorithms

Genetic algorithms (GA) are a technique for searching for solutions to a problem in an intelligent and efficient manner, modeling aspects of neo-Darwinian evolution. Here we will present a simplified description of how a GA works. The reader is referred to [7] for a more complete description.

Neo-Darwinian evolution explains adaptive change of species over time. Populations of individuals are subject to a process called natural selection. This process favors the group that is best adapted to its environmental conditions.

Adaptation has three aspects: heredity, variation, and selection.

Individuals, from an evolutionary point of view, have one function: to reproduce. Reproduction is the transmission of hereditary material from parent to offspring. It is the mechanism by which the information a population learns about how best to survive persists over time. Individuals that are better suited for survival are more likely to pass their genetic material on to the next generation.

Individuals are made up of sets of traits, called genes. Genes are the unit of heredity. The total amount of hereditary information an individual has is called its genotype. The manner of response in an environment and physical embodiment of the individual is the phenotype [1]. The phenotype is the collection of expressed genes; not necessarily all genes are included in this set. This differentiation is important. Individuals carry some information that may not at the moment improve their ability to survive, but the information is retained in the population's memory, perhaps to be used at a later time. Further, the amount of information a population knows increases over time, allowing individuals to live in a greater amount of environmental conditions.

The genetic material received by offspring due to reproduction is a unique set. The phenotype varies in small, seemingly undirected ways. The differences are important because they may give the offspring some critical advantage at some point. This increases the chance that the offspring will survive and reproduce.

Selection is an operation on the population. The individuals least suited for survival are likely to perish, unable to reproduce. If the environment is stable, and the population is well adapted, the population will grow large. There will be more individuals competing for resources. Individuals who are better able to exploit those resources are selected to survive and reproduce.

A genetic algorithm models natural selection. A population of individuals, representing solutions to a given problem, are evolved over a discrete number of time steps. All individuals in the population at one time step are evaluated and given scores; this is called the fitness. The fitness function is the most important requirement of a genetic algorithm: it must be able to identify correct, incorrect, and partially correct solutions. Over time, there are an increasing number of individuals who have more correct partial solutions. Those with the highest fitness are selected and reproduce a number of times proportional to their fitness. During reproduction, two (or more) operations are applied to the individual; usually these are crossover and mutation. Other operators are described in [7] and [1]. Crossover is the swapping of pieces of information between two different individuals. Mutations are random changes made to a single individual. Crossover and mutation only occur to an individual if certain probabilities are met; some individuals are copied into the next population without change.

Individuals are typically represented as bit strings. The position of the bit in a string is an encoding for a part of the solution. In a function optimization problem, for example, some function may take eight different parameters. If three bits could encode all possible parameter values, then each individual would be 24 bits long. Mutation and crossover are easy to implement with this representation.

A common way at looking at what kind of problem a GA solves is a fitness landscape. A landscape is a hyperplane of n dimensions, where n is the number of parameters. For some problems, this landscape is described as "hilly". More fit solutions are higher up on the hills. Genetic algorithms search these hills for optimal solutions.

Suitability of GAs to this problem

This problem is potentially very difficult for a genetic algorithm. The fitness landscape of solving the "parse a piece of text" problem is viewed as a large spike. This is true if one views the fitness function as merely "did the individual parse the text, or not?" A genetic algorithm performs poorly in such an environment.

We suggest a different view. While the fitness landscape for parsing some text as a whole is unsuitable for GAs, the fitness landscape for building parse trees for one sentence is not as intractable. There are many trees that could be built from one sentence. The notion of partial correctness is critical in a GA. The algorithm must be able to identify individuals who solve at least part of the problem. We have created a fitness function that will be able to give higher scores to individuals based on how much of a given sentence they parse. Since GAs implicitly search many possible

solutions at the same time, we feel the GA would be able to induce a grammar, given enough time.

Further, we cannot expect to be able to start the experiment by giving the algorithm full-length complex sentences. It is important to train the algorithm slowly. With this in mind, we will use some ideas from developmental psychology. More information can be found in [Barbara's paper]. The basic idea is that children learn syntax slowly and in a certain order. We feel that the algorithm must be introduced to syntactic features in a similar, slow order.

To ease the difficulty of finding and keeping a correct grammar, we introduce a few extra components that we believe will assist the GA. These include an intelligent guessing structure, a method of learning in each generation, a method to transfer what one has learned to one's offspring, and specialized genetic operators.

Our Approach

Parser and Tagger

Two components necessary to the experiment are the parser and the tagger. The parser, given a grammar, determines whether a given sentence is grammatically correct. There are many kinds of parsers, all with their respective strengths and weaknesses.

The simplest parser is top-down recursive. Beginning with the start symbol, the parse tree is built by recursively expanding symbols on the right hand side. Search usually is performed left to right, i.e. the first instance of a left hand side is searched fully before any other rule for the same left hand side. The problem with top-down recursive parsing is that the same trees are often searched repeatedly. Simple bottom-up recursive parsing is similar to top-down; it begins at the terminals and recursively builds trees.

More complex bottom-up parsing, like LR, are computationally expensive. For each grammar, a set of LR items are created. LR parsing searches more than one path at once. Each LR item consists of a pointer to where the parse is currently situated, what symbols are next on the input, and possible rules that could apply.

For the first experiment we will use hand tagged sentences for training and testing. Once we prove that GAs can learn the syntactic structure of a language, albeit slowly, we will increase the complexity of the tags used. Complex tags are necessary to create rules for more difficult language structures, like relative clauses. At this point we would use a tagging component, which likely will consist of a parser. This parser is not necessarily the same parser as the one that is used to evaluate individuals. The evaluating parser must be simple enough to allow for relatively quick parsing of sentences given an individual's grammar. The parser used by the tagger can be arbitrarily complex.

Representation of Individuals

Each individual in a given generation must represent the rules which have come, over the evolution of that individual, to produce maximal success in parsing. The rules must be represented in a manner conducive to the genetic operators at work as well as to other components of the proposed system. This especially includes support for the constituent guesser, which creates new rules. Moreover, rule representations must support whatever mechanisms are introduced to handle grammar isomorphisms, which complicate both representation in general and the genetic operators in particular. The following addresses the general problems of grammar equivalences and isomorphisms and discusses our proposed solutions. Equivalent, in this description, means weak equivalence, while isomorphic refers to strong equivalence.

Grammar equivalences occur because for any given language there is a potentially infinite set of grammars which derive precisely the same set of sentences. Grammar isomorphisms occur because context-free grammars are processed strictly as production rules, without any regard to semantics. For example, consider the following individuals:

G1: (abc -> ghi def), (abc -> def), (def -> <noun>), and (ghi -> <art>)

G2: (qrs -> nmz), (qrs -> wlp nmz), (nmz -> <noun>), and (wlp -> <art>)

The rules from both individuals produce precisely the same finite set of derivations, "<art> <noun>", "<noun>",

using derivation trees that are structurally equivalent. These two grammars are isomorphic. Additionally, consider this individual:

G3: (abc \rightarrow lym), (lym \rightarrow rqs), (lym \rightarrow wnt rqs), (rqs \rightarrow <noun>), and (wnt \rightarrow <art>)

This grammar produces the same set of sentences, though it uses differently structured parse trees. G3 is equivalent to but not isomorphic with G1 and G2.

In the case of grammar equivalence nothing special need be done. It is precisely the point of a genetic algorithm to allow multiple individuals to develop very similar solutions to a problem. The genetic algorithm, using fitness measures, selects which individuals should propagate to the next generation. In the case of the grammar equivalence between the G3 and the isomorphic pair G1 and G2 the fitness measure favoring smaller grammars would tend to cause G3 to die out of the population after a few generations. Hence, multiple grammars that successfully parse sentences pose no problem simply because of the nature of the genetic algorithm. The fitness function referred to will be described in more detail below.

However, grammar isomorphisms do cause some problems. During the genetic operator of shallow crossover, single rules are taken from two individuals and swapped. This is done without regard to the other rules with which the chosen rules interact. So for instance, if G1 and G2 were selected for shallow crossover, rule 1 of G1 might be swapped with rule 2 of G2 (selected randomly). Without reference to the interaction of these rules with others from the grammar in particular derivations, how can we recognize that the two rules indeed do serve the same purpose in each grammar? Taken alone they merely indicate the replacement of one nonterminal with two other nonterminals, all of which are not known to be related in any way!

The problem is slightly more complicated for the genetic operator of deep crossover, in which an entire path from some root of the grammar to a terminal is swapped between two individuals. In this case we might be tempted to assert that isomorphic paths (with the same structure but different names) correspond to the same derivations for both grammars. It is possible for a sequence of nonterminals to have additional replacement rules present in one grammar but absent from the other. These would be overlooked by the above assertion.

To solve the problem of isomorphic grammars, we need some way to name constituent groupings and the hierarchically constructed nonterminals that are shared among all individuals. This can be done using a mapping structure that is global in the genetic algorithm and maintained between generations. Whenever a new rule is to be constructed, the table is searched for the symbols on the right hand side. If such a rule exists, then the name in the table for the left hand side is used for the left hand side of the rule in the individual. If the rule is not there it is added to both the table and the individual with whatever name is generated for it. This name is guaranteed to be unique in the map. This same procedure is applied for sequences of nonterminals or both terminals and nonterminals. This ensures that when a rule like (abc \rightarrow ghi def) is copied into an individual, those nonterminals can be resolved with respect to the individual's current set of productions.

With those ideas in mind, we can choose a representation for the individual's rules. We chose a pair whose first element is the left hand side of the rule and whose second element is a list of all the symbols on the right hand side of the rule. Operations on representations are described below.

Whenever a constituent grouping occurs for which there are no rules in an individual, some pattern matching algorithm must be used to choose a new rule to build. The details of this rule guessing mechanism follow shortly. We can keep certain information within each individual to aid the rule guesser in its task. This information consists of preference values for existing rules. This information is unique to each individual, but applies to the rules which are stored in the global map. Therefore each individual can store pairs whose first element is the map index for certain constituent/nonterminal groupings and whose second element is a use-count for that rule within that individual. Thus, one individual may have (abc \rightarrow <art> <noun>) paired with a preference of 5 while another may have the same rule paired with a preference of 3. This reflects that the rule has been successfully applied more frequently in the first individual than the second. Such information, as shall be seen, can make the rule-guesser significantly more intelligent and should lead to faster convergence. The first diagram in the appendix demonstrates these representational choices in the context of the entire population.

Alternatively, we can keep different values in the preference list of the individuals. These preferences are defined with

respect to the structure of the constituents. As an example, given a partially parsed sentence “<det> <noun> vp,” the individual may have a high preference for grouping the first two elements into one constituent, then group this new constituent with the last. Or the individual may group the last two constituents together. Preferences, when coded in this fashion, may be subjected to genetic operators as well.

Fitness Function

Fitness functions are used in genetic algorithms to evaluate each individual for inclusion in the next generation. The fitness landscape refers to the multidimensional surface of the fitness function over the possible individual configurations. Normally genetic algorithms are only effective if there is some smoothness to the fitness landscape, populated by many local minima and maxima of which there is at least one relatively high maximum. Fitness functions are often composed of a weighted combination of several different measures of an individual.

Some of the measures that are useful in this application include, in decreasing order of significance, the successful parsing of a sentence using an individual’s grammar, a relatively small number of rules in an individual’s grammar, and a relatively small average length of the right hand sides of an individual’s rules. The only difficulty with these measures is that the most significant contribution comes from an evaluation that is binary. Successful parsing of a sentence is a discrete, yes-or-no question. This leads to a fitness landscape with a single large spike in the center. Some individual would have to start out very near this spike, which is unlikely with grammar inference unless the priming of individuals includes nearly complete grammars, which contradicts the idea of using a genetic algorithm in the first place. We need to develop a means to smooth the fitness landscape in order to transform the discrete question about successful parsing into a continuous question about the quality of the parse.

One technique for achieving this transformation and smoothing the fitness landscape from a spike to a smooth surface with multiple local extrema may be called the “Area Method”. As sentence complexity grows more rules are needed for a successful parse. If we count the number of different rules used in a partial parse as a function of the number of words parsed, we have a kind of area formed by the product of the used rule proportion and the consumed word proportion. This gives a rough “quality of parse” measure. Of course, this simple evaluation can be misleading. Like most contributors to fitness, it is added into the evaluation in a weighted fashion as is, without concern for its flaws. The point of a combined fitness function is to trust the contribution of each measure and its weight, allowing other measures to make up for the flaws in one particular measure. This is probably not sufficient alone, but does help move the evaluation away from the strictly discrete question of “did it parse?”

Another technique for smoothing the fitness landscape derives from the preferences stored in each individual. If these values are propagated, as suggested, up through the grammar so that we have an indicator of the frequency of use of each rule in an individual, then we can use that information during an evaluation parse to indicate whether the grammar was used in a fashion commensurate with its past successful use. The idea is that some individual’s grammar may parse a sentence but only by using several obscure rules repeatedly and almost never using much more common and general rules. Sometimes this might indicate that this individual is less suited for parsing the sentence than some other individual might be. Again this heuristic could also be misleading, which is why it is important that it is only one contributor to the fitness among many. The right rule-use frequency relative to the preferences in an individual will have to be determined empirically in the early development of the genetic algorithm.

Finally, we are investigating a technique we call LR-smoothing. LR-parsing is a table-driven shift-reduce parsing algorithm. To use it, one takes a grammar for a language and generates an LR parse table. Along the way one must produce a huge collection of data structures called LR items. Each LR item represents, essentially, how far along the current parse is in each of the rules that might apply. Our LR-smoothing technique assigns a score to each LR item without actually constructing it (thus reducing the computational load so that it is feasible to do this for each individual in each generation). It asks the questions used to construct the items and uses the answers to assign the score. During parsing, when a rule is used the score for reduction by that rule at that point of the parse is added to the overall measure for the parse. The final value is normalized and we have essentially a percentage parse. We hope this will significantly change the discrete spike in the fitness landscape—not only to smooth it but also to introduce additional extrema and give the genetic algorithm a better chance to succeed. This should accelerate convergence. As

with any genetic algorithm, the design of successful fitness functions is a complex art. Further empirical results will no doubt guide the continued development of fitness functions for grammar inference.

Genetic Operators

We define two genetic operators for this experiment: shallow crossover and deep crossover. Normally in a genetic algorithm one uses another operator, mutation. Mutation is difficult for this problem; this is due to the use of the mapping structure described earlier. We cannot simply change symbols on the left or right hand sides of rules, since such changes would violate the information found in the map. Experimentally, mutation could change the ordering of symbols in the right hand side of some randomly selected rule. It is unclear whether this operator will help or hinder the algorithm.

Shallow crossover is the copy of exactly one production rule from one individual to another during reproduction. The major difficulty faced at this point is resolving the symbols present in the new rule with the remainder of an individual's grammar. We have two choices at this point: extended shallowness, or pure shallowness. The extended shallow operator performs the following actions: if all of the symbols in the production are unknown, look up the symbols in the map. With this table, an individual should be able to find a path of symbols that lead from some already existing rule to some symbol in the new rule. If there is more than one way to join the new rule with the existing grammar, the individual chooses at random a connecting path. The pure shallow operator, on the other hand, simply attaches the new rule to the grammar, with no rule resolution involved.

The deep crossover operator moves an entire chunk of rules between individuals. Beginning at the start symbol, one entire path down the rule tree is chosen at random. This entire path, consisting of numerous rules, is transferred to the other individual. Duplicate rules are removed. Since there is an entire path from start to terminals, no other symbol resolution is necessary.

The Guesser

Whenever an individual is presented with an unknown set of constituents a new rule must be constructed. In early generations these guesses may occur randomly. A small and random number of constituents are chosen in sequence and a new nonterminal is generated as the left hand side of a new rule whose right hand side consists of the chosen sequence of constituents. As the individual matures through the generations certain patterns will reoccur. Such recurrence can give clues about better guesses for new rules. Furthermore, we can use Bayesian inference techniques. Bayesian inference is a method by which uncertainty values propagate upward through implications in predicate calculus. In our situation productions serve as these implications. Preference would be given to certain groupings of nonterminals and terminals as well as to groupings composed entirely of terminals.

For example, consider the following sentence. "The thief, who we never saw, took some precious stones." One possible part-of-speech tagging that we might expect to accompany this sentence is "<art> <noun> <relpn> <noun> <adv> <verb> <verb> <adj> <adj> <noun>." Note that the commas were ignored; if we used a tagger that identified those we may use them in the grammars of individuals to produce higher quality parses. Assume that the individual being trained on this sentence has already seen the rules (np - > <art> <noun>), (np - > <adj>*<noun>), and (vp - > <verb> np). The rule (np - > <adj>* <noun>) is using the * for the Kleene closure operation in regular expressions to indicate concisely the more verbose context-free form of the same pattern. From those rules a partial parse tree can be built that covers all but the relative clause "who we never saw". The second figure in the appendix illustrates this.

So the rule guesser must come up with new rules to parse the remaining tokens. One way the preferences stored in each individual can help is by indicating what not to choose. For example, suppose that a predecessor of this individual in an earlier generation had randomly chosen to pair the "<relpn> <noun>" sequence as a constituent. Such a rule would be unlikely to have a very high preference (use-count from application in a successful parse). Indeed it will likely die out of the individual's progeny in short order. But at the moment it is present in the individual with a very

low preference. The rule guesser might choose to try any other random combination but this one because of its low value.

Another way the preferences stored in each individual can help the rule guesser is by identifying related rules whose right hand sides match part of the pattern. New rules can be made by modifying the right hand sides of those rules. For example, in the sentence from the above illustration we have the subsequence “<noun> <adv> <verb>” inside the relative clause that has no applicable rules. Suppose that the rule (s17 - > <noun> <verb>) is in the individual with a high preference count. The rule guesser uses pattern matching to identify that the tokens in the right hand side bracket the noted subsequence and generates a new rule such as (s17 - > <noun> <adv> <verb>). While this may not be an ideal rule, it and various others like it will be generated by different individuals and then the prowess of the genetic algorithm can be trusted to isolate the grammar with the most useful version.

Intelligent rule-guessing is just one of several aspects of this genetic algorithm that essentially involve micro-evolution of the generations of individuals inside the larger evolution that is the work of the primary genetic algorithm itself. The motivation for these deviations from standard genetic algorithms is to provide faster convergence in a high-dimensional fitness landscape for generations of individuals whose evaluation is computationally intensive to begin with.

The Process

The experiment shall proceed as follows. First, an initial population of individuals is made. All have empty grammars and randomly assigned preferences. The entire population must be evaluated so selection can occur. Some finite number of training sentences are chosen from the sample set. This sample set, in the beginning, will be hand tagged sentences with increasingly complex syntactic structure. Some finite number of test sentences are chosen as well. These sentences should exhibit the same syntactic structure as those in the training set.

We may seed the individuals with simple concepts of grammar, like the structure of wh-question sentences, or imperative sentences. Doing so may cause the population to converge more rapidly. Minimally it will allow some individuals to be nearer the major correct landscape spike.

When an individual is evaluated, first it is shown the set of training sentences and learning commences. During the training period, any time an individual is parsing a sentence but is unable to complete the parse (i.e. when a sentence contains new syntactic structure) the individual is allowed to create new productions. The decision of where to form constituents is determined in two ways. First, structural similarities between the test sentences are searched, and new rules are created to define those similarities. Second, the individual may simply have to guess. Using internal preferences, the individual uses the process described above to create new rules. Any new rules are recorded in the symbol table. New symbols for the left hand sides are generated by the symbol table component; no clashes in symbols will occur.

Learning ceases for an individual in a particular generation when it has successfully parsed the training sentences, or after a certain number of tries have occurred. The test sentences are then applied to the individual in conjunction with the fitness function. The individual is scored and the next individual is evaluated with the same training and test sentences.

Over time the entire population will learn an increasing amount of the syntax of the chosen language. At some point it should have enough knowledge of the domain that it can parse entire texts.

Conclusion

In this paper we have presented the components and process of a machine that, given some bits of natural language, should induce the structure of the sample. Note that at this time no prototypical code has been written. Please understand that this is simply an experimental proposal. We welcome any feedback the reader may have concerning the reality of the computational difficulty of the problem.

References

- [1] T. Back. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, New York, 1996.
- [2] G. Flake. *The Computational Beauty of Nature*. MIT Press, Cambridge, 1999.
- [3] K.S. Fu and T.L. Booth. Grammatical inference: Introduction and Survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 8:343-375, 1986.
- [4] D. Jurafsky J.H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. Prentice Hall, New Jersey, 2000.
- [5] J.R. Koza. *Genetic Programming: On the Programming of Computers by means of Natural Selection*. MIT Press, Cambridge, MA, 1992.
- [6] M. Lankhorst. *A Genetic Algorithm for the Induction of Context-Free Grammars*.
- [7] M. Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, Cambridge, 1996.
- [8] P. Wyard. Context free grammar induction using genetic algorithms. In R. Belew and L.B. Booker, editors, *Proceedings of the Fourth Conference on Genetic Algorithms ICGA '92*. Morgan Kaufmann, 1992.

Appendix

Representation of Individuals Sample Guessing