Setting Language Parameters
Using Genetic Algorithms

by
Juliet C. Bourne

Submitted to the Department of Electrical Engineering and Computer Science in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 1992


The author hereby grants to MIT permission to reproduce and to distribute copies of this thesis document in whole or in part.

Author

Department of Electrical Engineering and Computer Science
May 18, 1992

Certified by

Robert C. Berwick
Associate Professor
Thesis Supervisor

Accepted by
Leonard A. Gould
Chairman, Department Committee on Undergraduate Theses
Setting Language Parameters

Using Genetic Algorithms

by

Juliet C. Bourne

Submitted to the Department of Electrical Engineering and Computer Science on May 18, 1992, in partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science and Engineering

Abstract

Although a genetic algorithm is most likely not the way language is acquired by humans, genetic algorithms provide a way for a computer program to "learn" a natural language. Given the Principles and Parameters theory in linguistics, the vital information about a natural language can be encoded as parameter settings. A genetic algorithm can search for the parameter settings for a particular language, given input sentences from that language. Once the parameter settings are found, the program is well on the way to being able to understand the language. In my thesis, I present a very simple example of how a genetic algorithm can be used to search for parameter settings of English. Only word order parameters are used and they are kept necessarily simple, sacrificing linguistic reality. The genetic algorithm itself is also kept very simple, only using reproduction, mating, and mutation. The results of running the genetic algorithm on input sentences are mixed. The genetic algorithm itself works well, while randomness, the input, and the choice of parameters cause multiple solutions to be found, where theoretically there should only be one. A better parameter system and better input is needed to solve the problem of having multiple solutions for a single language. I think the results show that the overall concept of using a genetic algorithm to set language parameters works and that it is the linguistics that needs more work.

Thesis Supervisor: Robert C. Berwick
Title: Associate Professor
Acknowledgments

This thesis is dedicated to everyone who has asked me what my thesis is about for the last seven months. I hope this explains it better than the short answer I gave when you asked.

I would like to thank: Bob Berwick, my academic and thesis advisor, for all of his help during this past year – including providing the topic of this thesis! Steve Wadlow, a very good friend, for lending me moral support and an account on psyche.mit.edu which gave me access to the Childes database. Matt Power, a fellow SIPB member, for reading the rough draft and providing helpful feedback. All of the professors of my linguistic and cognitive science courses, for piquing my interest in natural language acquisition and the theory of Universal Grammar. My mother, for keeping on my back so I didn’t sell myself short. Everyone at SIPB and Random, for just being there. And George Christopher, for sanity-checking the final copy, for listening to me stress over everything, and for the flowers.
Contents

1 Introduction ................................................................. 7

2 Theory ................................................................. 12
   2.1 Principles and Parameters ............................................. 12
      2.1.1 Word Order Parameter Theory ..................................... 14
      2.1.2 Word Order Parameters Used .................................... 16
   2.2 Genetic Algorithms .................................................. 17

3 From Theory to Practice ................................................ 18
   3.1 Implementing the GA .................................................. 18
      3.1.1 Individuals ..................................................... 18
      3.1.2 Fitness .......................................................... 19
      3.1.3 Reproduction ................................................... 19
      3.1.4 Mating .......................................................... 20
      3.1.5 Mutation ....................................................... 20
      3.1.6 The Zeroth Generation .......................................... 21
   3.2 The Parser ........................................................... 21
   3.3 The Input ............................................................ 22
   3.4 The Lexical Database ................................................ 23
   3.5 Overcoming Randomness .............................................. 23
   3.6 evolve ............................................................... 23

4 Results and Conclusions ............................................... 25
List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1-1</td>
<td>Flowchart of the genetic algorithm used in this thesis</td>
<td>9</td>
</tr>
<tr>
<td>1-2</td>
<td>An example run</td>
<td>10</td>
</tr>
<tr>
<td>2-1</td>
<td>The structure of a verb phrase under X-bar theory</td>
<td>15</td>
</tr>
<tr>
<td>2-2</td>
<td>The structure of a V&quot; phrase</td>
<td>15</td>
</tr>
<tr>
<td>2-3</td>
<td>The template grammar</td>
<td>16</td>
</tr>
<tr>
<td>3-1</td>
<td>Cross-over of two individuals</td>
<td>20</td>
</tr>
<tr>
<td>3-2</td>
<td>An example run with a random walk</td>
<td>24</td>
</tr>
<tr>
<td>4-1</td>
<td>An example of an overgeneralized solution</td>
<td>26</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

Natural language acquisition, that is, the way humans learn their native language, has been the focus of much linguistic and cognitive science research. The ease and uniformity with which children acquire language is remarkable, suggesting that some part of language is "hard-wired" into the brain at birth. The search for this part of language which is "universal" has led to the Universal Grammar theory, which is parameterized. This theory of a Universal Grammar provides an opportunity to develop a "natural" way for a computer to learn a language. Natural language acquisition probably does not proceed in the "survival of the fittest" way a genetic algorithm searches for the most optimal solution to a problem. However, a genetic algorithm provides a good way to search for an optimal or near-optimal solution in a large search space. The parameters in the Universal Grammar theory provide such a space, with a different optimal solution for each language. Thus, using a genetic algorithm to search for the parameter settings of a language seems like the best way to wade through all of the possible parameter settings for a language. Once the parameter settings for a particular language are found, a computer can understand, or at least parse, that language.

In linguistics, the Principles and Parameters theory proposes that all natural languages are based on the Universal Grammar (UG), which is "hard-wired" into the brain. Variation between languages comes from the parameterization of the UG. For instance, word order varies from language to language, i.e. the settings of parameters
which determine if the subject of a sentence follows the predicate, if the determiner comes before the noun in a noun phrase, et cetera, vary from language to language. When children are exposed to their native language, their language acquisition device (however it may be manifested in the brain) is triggered by the presence of certain grammatical constructions. It sets the parameters corresponding to these constructions to the appropriate values. In this fashion, it is theorized, all of the parameters in the UG are set and children “learn” their native tongue.

Genetic algorithms (GAs) do not fit this model of learning a language, but nevertheless they provide a method for finding the correct parameter settings of a language. GAs are modelled after the survival of the fittest model of evolution in biology. A search begins with a population of randomly generated individuals. Given a fitness test, the fitness of each individual is determined. The more fit an individual is, the more likely it is to reproduce and thus pass on its traits to the next generation. This fitness test simulates the competition for mates and resources, as well as the longevity of an individual. Mating and mutations provide variation of the “gene” pool so that new combinations of traits can be generated and tested. This process of evolution based on reproduction, mating, and mutation continues for generation after generation and, in this way, the optimal, or near optimal, individual (or solution) is found. In the case of natural language parameters, individuals are a series of parameter values. The fitness test would be the ability of the parameter settings, along with the corresponding principles, to parse sentences from the target language. Given input sentences from a natural language, a genetic algorithm can be used to search for the correct parameter settings for that language.

The purpose of my thesis was to write a program which would learn the word order of a language, given only natural sentences from that language. I used a simple genetic algorithm to search for the values of word order parameters for English. The parameters consist of the ordering of the right-hand side of binary grammar rules, such as “NP→AP N” or “NP→N AP” where NP means Noun Phrase, AP means Adjective Phrase, and N stands for Noun. In English, the former is the correct ordering. Individuals consist of numbers which represent the order of each phrase in
Generate random individuals for zeroth generation

Determine fitness of each individual

Convert parameter settings to grammar

For each sentence in corpus, give grammar and sentence to parser

Fitness = number of sentences successfully parsed

Does highest fitness exceed threshold?

no

Is there a "perfect" individual?

no

Max. generation reached?

no

Create gene pool based on fitness for next generation

Randomly mate individuals in the gene pool to create new individuals

Mutation generation?

no

Mutate one gene in one individual

yes

Run Bi-search

Return solution or fittest individual

yes

no

Figure 1-1: Flowchart of the genetic algorithm used in this thesis.
> (evolve 50 10 50 0.94)
Generation 1: Best=(0 1 1 0 0 0 0 1 1 0 1 0 0), 57% fit
Generation 2: Best=(0 1 1 0 0 1 1 0 0 1 0 0 0), 82% fit
Generation 3: Best=(0 1 1 0 0 1 1 0 0 1 0 0 0), 82% fit
Generation 4: Best=(0 1 1 0 1 0 0 1 0 1 1 0 0), 86% fit
Generation 5: Best=(0 1 1 0 0 0 0 1 0 0 1 0 0 0), 91% fit
Generation 6: Best=(0 1 1 0 0 0 0 1 0 0 1 0 0 0), 91% fit
Generation 7: Best=(0 1 1 0 0 0 0 1 0 0 1 0 0 0), 91% fit
Generation 8: Best=(0 1 1 0 0 0 0 1 0 0 1 0 0 0), 91% fit
Generation 9: Best=(0 1 1 0 0 0 0 1 0 0 1 0 0 0), 91% fit
Generation 10: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
(0 1 1 0 0 0 2 1 0 0 1 1 0 0)

Figure 1-2: An example run in which the correct parameter settings are found after 10 generations, with a generation size of 50 individuals, a mutation rate of one every 10 generations, and a brute force search used after the fitness of the best individual exceeded 94%.

the grammar. Thus each individual has a grammar associated with it. The fitness test consists of using an individual's implicit grammar to parse a corpus of natural English sentences. The parser that is used is very simple and just returns true or false, depending on whether a sentence could be parsed with the given grammar. The more sentences an individual's grammar can parse successfully, the more fit the individual is. Using the process of reproduction, mating, and mutation, the genetic algorithm finds the parameter settings representing a grammar which can correctly parse all sentences in the corpus. (See figure 1-1.)

For the most part, the general concept worked. (See figure 1-2.) A minor problem was encountered with the randomness in the genetic algorithm. When a near optimal solution is found, or when the average fitness is high, the genetic algorithm becomes a "random walk" around the optimal solution and the fitness does not increase quickly (see section 3.5). It could take hundreds of generations for the last parameter to click into place finally. Thus, a brute force search can be used to finish up the search process once the genetic algorithm has found a near optimal solution. Another problem was caused by the "free order" parameter setting. A free order parameter means that a language allows both orderings of the right-hand side of a binary grammar rule. In
English, the only free ordering is that of a verb and an adverbial phrase. However, even though adverbial free ordering is questionable, other natural languages do have free ordered phrases and thus are necessary for a general theory of parameters. Having free ordered phrases not only increases the time it takes to parse a sentence (since two orderings must be tested for a phrase), but it also overgeneralizes parameter settings. Thus, with incomplete input, there may be more than one grammar which can parse the corpus of input “perfectly” because free ordered phrases can interact in such a way as to parse a given sentence successfully, but not correctly in terms of English syntax. To solve this problem, either the ability to have free ordered phrases must be eliminated (which is not a linguistically realistic solution) or the input sentences must be specific enough to rule out “perfect” overgeneralized parameter settings.

After overcoming the above problems, the genetic algorithm was able to find the correct parameter settings remarkably well. Due to the randomness inherent in the genetic algorithm, the time (or number of generations) required to home in on the solution varied. However, a solution was usually found within 100 generations\(^1\), using the brute force search once the most fit individual exceeded a given fitness level. Overall, given the results of this simple system, it seems feasible to use genetic algorithms to enable a computer to “learn” a language. The biggest problem facing the use of GAs to find the correct parameter settings for languages is that not all of the parameters (or principles, for that matter) that define languages have been formalized and agreed upon. Thus, until a complete theory of Principles and Parameters is reached, GAs can only be used to learn certain features of natural languages.

\(^1\)For the sample runs included in the appendix, if a solution was found, it was found with an average of about 30 generations.
Chapter 2

Theory

2.1 Principles and Parameters

If the child is viewed simply as a *tabula rasa*, as having no predisposition to analyzing data in terms of one formal system rather than any other one, then it is difficult or impossible to explain how the child does in fact come to acquire his first language within a relatively short period of time. [1, Comrie, pp.2-3]

The theory of Universal Grammar in linguistics explains the ease with which children acquire their native language and also formalizes the similarities and differences between languages. Without some innate ability to acquire language, it seems hard to believe that children could acquire their native tongue so rapidly and uniformly. Somehow, children exposed to the same language acquire the same grammar even though they are given different input. For example, one adult English speaker has basically the same descriptive grammar as another. Even at an early age, children have knowledge of complex linguistic rules such as the Binding Principle which was never taught to them explicitly. The Binding Principle determines what antecedents are allowed for a pronoun or a reflexive. Children know that in the sentence “He told him to go away” the pronoun “him” cannot be bound to the pronoun “he”. That is,

---

1As opposed to prescriptive, which is what an English class teaches.
“him” and “he” cannot refer to the same person because a pronoun, like “him”, must be unbound in its governing category, which in this case is the whole sentence. Thus, some part of language must be innate to be able to explain the uniformity and the ease of language acquisition, especially in the absence of explicit teaching. However, it is easy to see that a specific language in its entirety is not innate, since there are many different languages in the world and children are not constrained to learn only the language of their parents and no other. Indeed, children learn whatever language\textsuperscript{2} they happen to be exposed to while they grow up, which could be very different from the language of their parents.

The Universal Grammar theory also helps explain the similarities and differences among natural languages. For instance, the Binding Principle mentioned above is parameterized over the governing category in the UG. For English, the governing category of a pronoun or reflexive is the minimal clause with a subject, which is usually just the whole sentence. Thus, pronouns must be free (unbound) in their governing categories, while reflexives, such as “himself”, must be bound in their governing categories. For example, in the sentence “John asked me to call him”, the antecedent of “him” can be “John” because “John” is not in the governing category of “him” and thus “him” is free within its governing category. The governing category for “him” is the infinitive (untensed) clause “me to call him” where “me” is the subject. In Icelandic, however, the governing category for a pronoun is the minimal enclosing clause having tense. So in the equivalent Icelandic sentence to “John asked me to call him”, the pronoun “him” cannot refer to “John” as its antecedent since the minimal clause with tense is the entire sentence and “him” must therefore not be bound to a noun phrase within the sentence. The Binding Principle is the same for both English and Icelandic, however the value of the governing category parameter is different and explains exactly how binding in English is different from binding in Icelandic. (See [3, Wexler].) Thus, in this way, the Principles and Parameters theory is able to formalize the variations among languages.

\textsuperscript{2}Or even languages, if they are in a multilingual environment. Explaining how multiple languages are acquired simultaneously is beyond the scope of my thesis.
2.1.1 Word Order Parameter Theory

One of the main variations among languages is word order. The Principles and Parameters theory for the word order of languages involves many parts. Not only does the "headedness" of phrases play a part, but also the thematic roles of words contribute more information to the final word order of a language. A phrase can be head-initial or head-final, meaning that the main part of a phrase (i.e. the verb in a verb phrase) either at the beginning or the end of the phrase. For example, in English, phrases are mainly head-initial; the verb is first in a verb phrase, the preposition\(^3\) is first in a prepositional phrase, etc. The structure for phrases is described by X-bar theory and the headedness of phrases is determined by parameters which indicate head-initial or head-final. Word order also comes from the thematic roles of words. Verbs are subcategorized according to the arguments they take. For example, the verb "give" takes as arguments a "patient" and a "beneficiary". In the sentence, "John gave Mary the dog", "the dog" is the patient, or the object being given. "Mary" is the beneficiary, or recipient of the object being given. "John" is the "agent" of the sentence, or the one doing the action described by the verb phrase. "Agent", "patient", and "beneficiary" are all thematic roles. The order of the arguments to a verb are usually fixed, in order to avoid confusion as to which is the patient, etc\(^4\). If the order of the arguments to the verb "give" were not fixed, the above sentence would be syntactically (and semantically) ambiguous since "the dog" could be the beneficiary and "Mary" could be the "patient". X-bar theory, thematic role theory, and several other theories all help determine the word order of a language.

X-bar theory is perhaps the simplest to understand: a phrase is simply a head with its arguments. The arguments are all on one side of the head or another. Thus, the grammar consists of a set of binary rules and the parse trees it generates consist of binary branching nodes\(^5\). For example, a verb phrase could look like figure 2-1. The

---

\(^3\)It’s called a preposition because of its position in the phrase. Other languages have postpositional phrases.

\(^4\)In languages such as Japanese and Korean, word order is much freer, but special tags indicate the thematic roles of words in the sentence.

\(^5\)Berwick, 6.863, Spring 1992
Figure 2-1: The structure of a verb phrase under X-bar theory.

Figure 2-2: The structure of a V’ phrase, which includes a verb, its argument, and its complement.

bar above the V stands for "phrase" and indicates that the verb is the head of the phrase. This can also be denoted by V’. Double-bar phrases (also denoted with double quotes, i.e. V”) indicate complements to the single-bar phrases they dominate. The complement of a verb phrase, for example, could be an adverb. Thus, the V” structure that appears in figure 2-2 consists of the argument and the complement to the verb. The head of the V” is the V’ and the argument is the adverb which complements the V’. In this way, the word order of a language can be specified through the headedness of each of the phrases in the binary grammar. Of course, a binary grammar cannot give the order of multiple arguments of verbs. That must be done through the theory of thematic roles and other principles. But a parameterized X-bar theory can give the general framework of the word order of a language.
Figure 2-3: The template grammar from which word order parameters are derived consists of 14 phrases. C stands for Complement, e.g., “what”. I stands for Inflection. Aux stands for Auxiliary verb. VP is Verb Phrase, NP is Noun Phrase, D is Determiner, Adv is Adverb, A and Adj are Adjective, V is Verb, N is Noun, P is P-position, and Spec is Specifier (e.g., “very”). V+o substitutes for the full subcategorization of verbs and simply allows zero or one object for any verb. Parentheses around a phrase or part of speech on the right side of a grammar rule means that that phrase or part of speech is optional.

2.1.2 Word Order Parameters Used

In my thesis, I have used a set of binary grammar rules loosely modelled after the X-bar theory described above. I have ignored the contribution of thematic roles to word order, since thematic roles would greatly complicate the parser and system needed to interpret these parameters. Thus, the word order parameters in my system are merely the headedness of each of the phrases in the grammar: i.e., head-final, head-initial, or free order. Free order means that the head of a phrase can appear first or last. The only example of this in English is the position of an adverbial phrase with respect to the verb. Figure 2-3 shows the “template” grammar, upon which the parameters are based. “Head-initial” versus “head-final” for each phrase in a specific is defined relative to the headedness of the phrases in the template grammar.

Thus there are 14 word order parameters in my system, one for each phrase in the template grammar. For a given language, the majority of the parameters should
be the same, since languages are usually primarily head-initial or head-final. For example, English is mainly head-initial. However, the grammar that I have outlined is by no means linguistically realistic, so the head-initial vs. head-final parameter setting might not mean much in terms of the actual headedness of a natural language.

2.2 Genetic Algorithms

Nature uses a particularly well-known search and optimization algorithm: genetics and the survival of the fittest. Creatures everywhere undergo this process at one time or another. Good survival traits get passed on to future generations as the fittest mate and reproduce. Characteristics of the population change slowly with time as these traits win out over other less optimal traits. Genes provide the encoding of the traits. Competition among individuals for resources and mates provides a “fitness test” which determines which individuals’ traits are the most likely to survive to the next generation. Mating passes traits (as genes) on, in various combinations, to the offspring. Every so often, a mutation will occur which will alter a gene. If this mutation provides a good survival trait, it might be passed on to future generations.

Genetic algorithms in computer science follow this natural pattern of search and optimization. For some problem for which the optimal or near optimal solution is sought, an encoding of the relevant material that makes up a solution must be devised. This corresponds to the genes. Then a fitness test must be created that can determine the fitness of each proposed solution (individual). This corresponds to the competition among individuals in a population. Once the fitness of all the individuals in a generation are known, replicas of individuals are placed in a gene pool. The number of replicas of an individual is based on the fitness of that individual; the fitter the individual, the more replicas of it go into the gene pool. Then individuals in the gene pool are randomly mated and mutation, if any, is introduced. The resulting individuals constitute the next generation and the whole process of reproduction, mating, and mutation starts over. Eventually an optimal or near-optimal individual appears and the search is over.
Chapter 3

From Theory to Practice

Four main parts make up the whole system: the actual genetic algorithm, a parser, a lexical database, and a corpus of natural input sentences. One additional part can be used, namely a brute force search algorithm, to finish up the search for the optimal solution.

3.1 Implementing the GA

Implementing a simple genetic algorithm is very straightforward. As described in section 2.2, the first step is to devise a way to encode relevant information into individuals. Then, a fitness function or test is made to determine the fitness of these individuals. To move from generation to generation, the process of reproduction, mating, and mutation is carried out. The choice of the starting generation is mainly arbitrary, but there are some issues to consider.

3.1.1 Individuals

Individuals which comprise the population in this genetic algorithm are a set of numbers which indicate the order of each binary phrase in the grammar. (See figure 2-3 for the template grammar.) “0” means that the order the right-hand side of a phrase is the same as in the template grammar, “1” means that it is the opposite of the order in the template grammar, and “2” means that the order is free. Since there are 14
phrases in the grammar being used, and thus 14 parameters, an individual consists of
14 numbers from 0 to 2, i.e. the settings of each parameter. Each of these numbers
can be thought of as a gene which specifies a certain trait, where a trait is just the
order of a certain phrase. The order of the numbers follows the order of the phrases
in the template grammar, i.e. the first number indicates the order of the first phrase
in the template.

3.1.2 Fitness

To determine the fitness of individuals, the genes, i.e. parameter settings, in an
individual are used to create the corresponding grammar which the parameter settings
specify. This grammar is then given to the parser along with each of the sentences
in the corpus of English sentences. A count is kept of how many of these sentences
an individual’s grammar can successfully parse. The fitness of an individual is the
percentage of the sentences from the corpus that its grammar parsed successfully.

3.1.3 Reproduction

The gene pool for the next generation is created by determining the fitness of every
individual in the current generation. All of these fitnnesses are summed together to
represent the total fitness of the generation. For each individual, the number of
replicas to be placed in the gene pool is the fraction of the total fitness the individual
represents times the total number of individuals in a generation, rounded to the
nearest integer.

Occasionally, due to rounding errors, the above algorithm for creating the gene
pool results in either too few or too many individuals for the next generation. Thus
individuals are removed from or added to the gene pool at random to produce a gene
pool with the right number of individuals.
LOCUS = 5

MOTHER: 1 1 1 1 1 1 1 1 1 1 1 1
FATHER: 0 0 0 0 0 0 0 0 0 0 0 0

OFFSPRING1: 0 0 0 0 0 0 1 1 1 1 1 1 1 1
OFFSPRING2: 1 1 1 1 1 1 0 0 0 0 0 0 0 0

Figure 3-1: The cross-over of two individuals creates two new individuals, each of which have traits from both parents.

### 3.1.4 Mating

Once the gene pool for the generation is ready, mating is done randomly among individuals in the gene pool. In the algorithm I use, mating consists only of cross-over. An even random number of individuals are paired up randomly and a locus for each of these pairs is chosen at random. The locus is a number between 0 and 13 which indicates at which location to split the two individuals. Two new individuals are created in a splitting and recombining process where each has the first part of one of the original individuals and the last part of the other individual (see figure 3-1). In this way, the genes in the gene pool are mixed around to create new combinations of genes.

### 3.1.5 Mutation

Every so often, mutation can occur. Mutation involves the changing of one gene in one individual. Although this seems like a very minor change in a large population of individuals, it can make a difference if it introduces a substantially beneficial trait. The choices of individuals and genes for mutation are random. The new value for a gene targeted for mutation is randomly chosen to be one of the two values which the gene currently does not have. It is best to keep the mutation rate low so that good combinations of genes are not destroyed randomly by mutation. In my implementation, the mutation rate can have different values for different simulations. The
mutation rate is a number indicating how many generation there are between each single mutation.

3.1.6 The Zeroth Generation

For the starting generation, the choice of individuals is random. That is, individuals are created by randomly choosing each of the 14 genes. The one restriction on this randomness is that the value “2”, or free order, is never given to a gene. Free order phrases are assumed to be rare, although they do occur frequently in certain languages. The presence of free order phrases slows down the parsing of sentences and creates overgeneralized parameter settings. Thus, all of the individuals in the zeroth generation have genes with only the values of “0” or “1”. However, mutation can still change a gene to have the value of “2”. With each mutation on an individual from the zeroth generation, there is a 50% chance that a “2” gene will be introduced into an individual since individuals from the zeroth generation only have “0”s and “1”s and the new value has a 50% chance of being a “2”.

3.2 The Parser

The parser that I have written for my thesis takes a grammar, which kind of phrase to parse, and an input sentence. It is a recursive descent parser¹ and returns true if it was able to successfully parse the input, otherwise it returns false.

The way the parser works is simple. It tries to match each word in the input to each terminal (such as Noun, Verb, etc) specified by the grammar. The parser determines the part of speech of a word by looking it up in the lexical database. It knows how to handle optional components of phrases, so if a terminal (or even a phrase) is optional and there is no matching word in the input it will ignore the optional terminal (or phrase). It also handles free-order phrases by trying both orders of a phrase before returning false or moving on to the next phrase, if the free-order phrase is optional.

¹A recursive descent parser is one which expands each phrase it encounters and recursively calls itself to try to parse each expanded phrase.
When trying to match a word in the input to the next possible terminal in the grammar, the parser will expand the next entry in the grammar if it is not a terminal. In order to avoid entering an infinite loop when parsing potentially recursive phrases such as a noun phrase, the parser keeps track of whether it is currently parsing an optional complement phrase (CP) or prepositional phrase (PP). If it is, it will not try to parse another such phrase which is optional. This rules out noun phrases such as “the dog in the house on the corner”, but without such a addition to the parser it would hang infinitely trying to expand the noun phrase in the prepositional phrase in the noun phrase, etc. Fully recursive phrases such as adjective phrases are not allowed to be recursive at all, and so phrases with two adjectives like “big blue block” are not allowed since such a phrase would involve a recursive expansion of the adjective phrase. Thus, although the parser is not completely capable of parsing recursive phrases, it suffices for the simple grammars and simple sentences that are used in my thesis.

3.3 The Input

The corpus of input sentences which is used as part of the fitness test for individuals is taken from the Childes database of natural child-parent interactions. In the Input appendix, the specific sources of sentences are listed. Only the parents’ utterances are used. The idea behind using natural utterances directed to children is to show that the genetic algorithm can acquire the parameter settings given the same input as children\(^2\). However, given any good variety of sentences, not necessarily “natural” sentences, the genetic algorithm should be able to set the parameters.

\(^2\)However, three sentences which do not appear in the database are used, although they are modelled somewhat after the sentences used from the database. These “unnatural” sentences are used to provide more variety than could be easily found in the database. For example, sentences which have noun phrase with its optional complement phrase in the subject position are not easy to find.
3.4 The Lexical Database

Once the corpus of input sentences was chosen, a lexical database including all of the words used in the sentences was constructed. The words were hand-tagged with parts of speech and placed into a binary tree. The parser uses this lexical database to find the parts of speech of the words in the input it is given.

3.5 Overcoming Randomness

One problem with genetic algorithms is that there is a certain amount of randomness. This randomness can cause the GA to perform a "random walk" around the optimal solution once it has found a near-optimal solution (see figure 3-2). It can sometimes take a long time before the GA randomly stumbles on the remaining parameter settings for the optimal solution. Thus, in order to avoid this random walk, a brute force search method can be used once the fittest individual in a generation exceeds a certain fitness threshold. This brute force method simply takes the fittest individual and changes each parameter setting. Any changes that improve the fitness of the individual are kept. In this way, an optimal solution is found.

3.6 evolve

The main procedure for the genetic algorithm I wrote is called evolve. It takes five arguments, two of which are optional. The first arguments is the number of generations to run the GA. If a solution has not been found by the time this number of generations have been run, the procedure will return the best individual and its fitness from the last generation run. The second argument is the mutation rate, representing the number of generations between mutations. The third is the generation size, that is, the number of individuals in each generation. The fourth argument is optional and indicates the fitness threshold above which to run the brute force search on the best individual. If it not specified, it defaults to nil and brute force search will never be used. The fifth argument is also optional. It provides the option of prohibiting
Figure 3-2: An example run in which the genetic algorithm performs a random walk around an optimal solution, namely (0 1 1 0 0 2 1 0 0 1 1 0 0). In this example, brute force search could have been used once it reached 95% fitness.

free order phrases, by specifying the number of gene values that are allowed. The default number is 3, meaning that a gene can have “0”, “1”, or “2” as a value. If this argument is set to 2, the corpus of input should reflect the fact that there are no free order phrases allowed. In English, this means that all adverbial phrases must appear on one side of the verb or the other.

See Appendix C for some results of running evolve.
Chapter 4

Results and Conclusions

Overall, the genetic algorithm managed to find sets of parameter settings which parsed the corpus of input successfully. The fact that it found parameter settings other than the ones expected which worked is mainly due to the inadequacy of the grammar, parameters, and input. The input was not specific enough, or did not provide enough positive evidence, to set the parameters uniquely. The grammar and parameter system used was too limited, in that it constrained the types of sentences that could be parsed. Had the grammar allowed richer grammatical constructions, it is possible that the input could have been chosen to provide enough examples to set the parameters uniquely. As it is, though, several other sets of parameter settings were found whose corresponding grammar could parse the corpus of input perfectly, but not necessarily correctly in terms of syntax. For example, the parameter which specifies the order of a verb phrase and its optional complement phrase could be set to “head-final”, meaning that the complement phrase would come before the verb, and the parameter which specifies the order of a noun phrase and its optional complement phrase could be set to “free”. While these are not the settings for English, since a complement phrase for a verb phrase comes after the verb phrase, the interaction between these two parameters is such that a sentence such as “The animals that Janna saw thought she had big eyes” could be parsed to have a syntactic structure other than the true English syntactic structure. Thus, this problem does not reflect on the genetic algorithm as a way to search for a solution; it only shows that the choice of encoding
Parameter Settings          Fitness
-------------------------  ------
0 1 1 0 0 1 2 1 0 0 1 1 0 0 95%
| 0 1 1 0 2 1 2 1 0 0 1 1 0 0 98%
| 0 1 1 0 2 2 2 1 0 0 1 1 0 0 100%

Figure 4-1: An example of how the interactions between parameters and the brute force search method cause overgeneralized solutions to be found.

of the relevant information is faulty and does not define clear solutions. Given the grammar and input it had to work with, the genetic algorithm did its best to find an optimal solution, and it usually did.

Randomness plays a major part in the genetic algorithm\(^1\) and thus the results reflect this randomness. The GA could find a solution within 10 generations, or it could fail to find a solution within 100 generations. The choice of the size of a generation\(^2\) and the mutation rate partially caused this, however the randomness in mating and mutation is the main cause since it controls how the genes are varied to create new combinations (after the zeroth generation). A “fit” combination could be stumbled upon in the first generation, or it could be found in the 99th. Once a near optimal solution is found, the randomness makes matters worse since the GA becomes a random walk around an optimal solution. The method for overcoming this, namely the brute force search used after certain level of fitness is achieved, did lessen the number of generations generally needed to find a solution. However, the brute force search often caused overgeneralized solutions to be found. The complex interactions among parameters favor the overgeneralized free-order parameter setting and since the brute force search changes one parameter at a time, from left to right, it usually produced overgeneralized solutions. (See figure 4-1.) The ability to have free-order phrases in itself causes overgeneralized solutions. For example, the parameter

\(^1\) How many times does “random” or a variation thereof appear in this paper?
\(^2\) The larger the generation size, the less generations it generally takes to find a solution, since the variation among a large number of individuals is greater and the chances that a near optimal solution is among these individuals is high.
settings "0 1 2 0 0 0 2 1 0 0 1 1 0 0" and the corresponding grammar is able to parse every sentence in the corpus. The free order setting for the Inflection Phrase, the phrase which determines whether the subject comes before the predicate, or *vice versa*, is superfluous. It includes both orders of the phrase and thus will parse the input correctly, always using the head-final order. So, solutions with superfluous free-order phrases were also found by the GA.

Despite these problems, I believe that the concept of using a genetic algorithm to find the parameter settings of a certain language works. Genetic algorithms are able to separate the "good" traits from the "bad" traits and propagate the good traits to future generations to combine with other good traits until the optimal solution is found. When the search space is large, this method of find a solution is better than a method that would have to consider each variable or parameter in turn. The problems that my program encountered were not due to any fault in the genetic algorithm method of searching. They were caused by the limited variety of input sentences and a poor choice of a parameter system, which allowed multiple solutions to exist in the search space. A more sophisticated parameter system would allow a greater variety of input sentences to be used, and both of these together should narrow the solutions in the search space down to just the one solution which corresponds to the parameters of the target language. All of the parameters that determine word order for all languages have yet to be formalized and thus developing a truly realistic parameter system is difficult. When such a parameter system is developed, the corpus of input sentences could theoretically include all of the parents' sentences in the English Childes database, given a sufficiently sophisticated parser. This would solve the problem of having input that does not provide enough variation since if children can acquire English from this input, so should the genetic algorithm. Thus, the system I have developed in this thesis is mainly constrained by the limitations of the linguistic theory used. I hope to continue working on the linguistic portion of the system, as well as exploring variations of genetic algorithms, in an attempt to develop a similar (but more sophisticated system) which can learn the word order of any natural language when given only natural sentences from the target language.
Appendix A

Code

A.1 Genetic Algorithm

(defun evolve (num mrate size &optional force (genes 3))
  (defun next-generation (current n)
    (format t "Generation `a: " n)
    (let* ((fit-list (mapcar #'(lambda (p) (calc_fitness p)) current))
            (i-max (most-fit current fit-list)))
      (format t "Best=`s, `-a% fit-%" (cadr i-max) (round (* 100 (car i-max))))
      (if (and force (< force (car i-max)))
        (bf-search (cadr i-max) (car i-max) genes)
        (if (= n num)
          i-max
          (if (= 1.0 (car i-max))
            (cadr i-max)
            (let* ((pool (adjust-size (reproduce current fit-list) size))
                    (offspring (mate pool)))
              (if (= 0 (mod n mrate))
                (next-generation (mutate offspring genes) (1+ n))
                (next-generation offspring (1+ n)))))
      (next-generation (random-generation size) 1)))

(defun calc_fitness (params)
  (defun cf-iter (corpus grammar successes)
(if (null corpus)
  successes
  (let ((s (car corpus)))
    (if (parse s '(cp) grammar)
        (cf-iter (cdr corpus) grammar (1+ successes))
        (cf-iter (cdr corpus) grammar successes))))

(float (/ (cf-iter *corpus* (params->grammar params) 0) *size-of-corpus*))

(defun reproduce (generation fit-list)
  (let (((fit-total (sum fit-list))
          (size (length generation)))
        (do* ((g generation (cdr g))
              (fl fit-list (cdr fl))
              (p (car generation) (car g))
              (num (round (* size (/ (car fl) fit-total)))
                    (if (not (null fl))
                        (round (* size (/ (car fl) fit-total)))
                        0))
              (pool (add-on p '() num) (add-on p pool num)))
          ((endp (cdr g)) pool))))

(defun adjust-size (pool size)
  (let ((len (length pool)))
    (cond ((< len size)
            (let ((lucky (nth (random len) pool)))
              (adjust-size (append pool (list lucky)) size)))
          (> len size)
          (let ((unlucky (random len)))
            (adjust-size (deleten unlucky pool) size))
          (t pool))))

(defun mate (pool)
  (defun m-iter (nlist new-pool)
    (let ((mother (car nlist))
          (father (cadr nlist))
          (if (or (eq father 'x) (eq mother 'x))
new-pool
(let* ((children (cross-over (nth mother pool) (nth father pool)))
       (p1 (replacen mother (first children) new-pool))
       (p2 (replacen father (second children) p1)))
       (m-iter (cddr nlist) p2))))
(let ((nums (append (select-random (length pool)) '(x))))
       (m-iter nums pool)))

(defun mutate (pool max)
(let* ((individual (random (length pool)))
       (gene (random *size-of-individual*))
       (new-individual (change gene (nth individual pool) max)))
       (replacen individual new-individual pool)))

(defun params->grammar (params)
  (defun p-iter (n params grammar)
    (if (null params)
      grammar
      (cond ((= (car params) 0)
        (p-iter (1+ n) (cdr params)
          (append grammar (cdr (nth n *template*))))))))
    ((= (car params) 1)
      (p-iter (1+ n) (cdr params)
        (append grammar (flip (cdr (nth n *template*))))))))
    ((= (car params) 2)
      (p-iter (1+ n) (cdr params)
        (append grammar
          (list (append '(free)
            (cadr (nth n *template*))))))))
    (t (error "Invalid param num: " (car params)))))))
  (if (not (= (length params) *size-of-individual*))
    (error "Wrong number of parameters: " "S --> " S" params (length params))
    (p-iter 0 params '())))

(defun cross-over (mother father)
(let* ((locus (1+ (random (- *size-of-individual* 2))))
(fhalf1 (butlast father locus))
(mhalf1 (butlast mother locus))
(fhalf2 (nthcdr (*size-of-individual* locus) father))
(mhalf2 (nthcdr (*size-of-individual* locus) mother)))
(list (append mhalf1 fhalf2) (append fhalf1 mhalf2)))

(defun change (gene individual max)
  (let ((new-val (random max))
         (old-val (nth gene individual)))
    (if (= new-val old-val)
      (change gene individual max)
      (replacen gene new-val individual))))

(defun flip (blist)
  (list (list (cadar blist) (caar blist)))))

(defun most-fit (generation fit-list)
  (defun mf-iter (mpair g ft)
    (if (null g)
      mpair
      (if (> (car ft) (first mpair))
        (mf-iter (list (car ft) (car g)) (cdr g) (cdr ft))
        (mf-iter mpair (cdr g) (cdr ft))))))
  (mf-iter (list (car fit-list) (car generation))
    (cdr generation) (cdr fit-list)))

(defun random-generation (size)
  (defun i-random (n)
    (if (> n *size-of-individual*)
      ()
      (cons (random 2) (i-random (1+ n)))))
  (defun g-random (m)
    (if (> m size)
      ()
      (cons (i-random 1) (g-random (1+ m))))
  (g-random 1))
;; auxiliary procedures

(defun sum (lst)
  (if (null lst)
      0
      (+ (car lst) (sum (cdr lst)))))

(defun add-on (elt lst num)
  (if (= num 0)
      lst
      (add-on elt (append lst (list elt)) (1- num))))

(defun select-random (max)
  (defun sr-iter (n lst)
    (if (= n 0)
        (delete-duplicates lst)
        (sr-iter (1- n) (append lst (list (random max)))))
    (sr-iter (random max) '())))

(defun replacen (n elt lst)
  (let ((len (length lst)))
    (if (> n len)
        (error "Index out of bounds! (~S > ~S)" n len)
        (append (butlast lst (- len n)) (list elt) (nthcdr (1+ n) lst)))))

(defun deleten (n lst)
  (let ((len (length lst)))
    (if (> n len)
        (error "Index out of bounds! (~S > ~S)" n len)
        (append (butlast lst (- len n)) (nthcdr (1+ n) lst)))))

;; globals

(setq *template* '(((CP ((opt C) IP-prime))
                    (IP-prime (IP (opt Aux)))))

32
\[
\begin{align*}
&(IP \ (VP-prime \ DP)) \\
&(DP \ ((opt \ D) \ NP-prime)) \\
&(VP-prime \ (VP \ (opt \ CP))) \\
&(NP-prime \ (NP \ (opt \ CP))) \\
&(VP \ (V-prime \ (opt \ AdvP))) \\
&(NP \ (N-prime \ (opt \ AP))) \\
&(V-prime \ (V+o \ (opt \ PP))) \\
&(N-prime \ (N \ (opt \ PP))) \\
&(AdvP \ (Adv \ (opt \ Spec))) \\
&(AP \ (Adj \ (opt \ Spec))) \\
&(PP \ (P \ DP)) \\
&(V+o \ (V \ (opt \ DP)))
\end{align*}
\]

(setq *size-of-individual* (length *template*))

(setq *target* '(0 1 1 0 0 0 2 1 0 0 1 1 0))

### A.2 Parser

(defun parse (input phrase grammar &optional pos stop-cp stop-pp)
  (if (not input)
    (all-opt phrase)
    (if (or (not input) (not phrase))
      nil
      (if (not pos)
        (let ((new-pos (lookup (first input))))
          (if new-pos
            (parse input phrase grammar
              new-pos stop-cp stop-pp)
            (error "UNKNOWN WORD -- \"S\" \(first input\))))
        (let ((elt (first phrase)))
          (cond ((or (eq elt pos) (memq elt pos))
              (parse (rest input) (rest phrase)
                grammar (lookup (cadr input))
                stop-cp stop-pp))
            ((optional elt)
              (cond ((eq (first-pos elt) \'cp)
                  \ldots))))))
  
  (cond ((eq (first-pos elt) \'cp)
    \ldots)))

33
(if (not stop-cp)
  (if (parse input
       (append
        (expand (first-pos elt) grammar)
        '((unstop cp))
        (rest phrase))
    grammar pos t stop-pp)
    t
  (parse input (rest phrase)
    grammar pos stop-cp stop-pp))
(parse input (rest phrase)
  grammar pos stop-cp stop-pp)))
((eq (first-pos elt) 'pp)
 (if (not stop-pp)
   (if (parse input
        (append
         (expand (first-pos elt) grammar)
         '((unstop pp))
         (rest phrase))
     grammar pos stop-cp t)
    t
  (parse input (rest phrase)
    grammar pos stop-cp stop-pp))
  (parse input (rest phrase) grammar
    pos stop-cp stop-pp)))
((parse input (append
    (expand (first-pos elt) grammar)
    (rest phrase))
  grammar pos stop-cp stop-pp))
(t (parse input (rest phrase) grammar
    pos stop-cp stop-pp)))
((eq elt 'free)
 (if (parse input (rest phrase) grammar
    pos stop-cp stop-pp)
    t
    (parse input (rest phrase)
      grammar pos stop-cp stop-pp))
    (parse input (rest phrase) grammar
      pos stop-cp stop-pp)))
  34
(let ((new-phrase (append
    (list (caddr phrase) (cadr phrase))
    (cddddr phrase))))
  (parse input new-phrase grammar
    pos stop-cp stop-pp)))
  ((unstop elt)
  (if (eq (cadr elt) 'cp)
    (parse input (rest phrase) grammar pos nil stop-cp)
    (parse input (rest phrase) grammar pos stop-cp nil)))
  ((not (terminal elt))
    (parse input (append (expand elt grammar) (rest phrase))
      grammar pos stop-cp stop-pp))))))))

(defun terminal (elt)
  (memq elt *the-terminals*))

(defun optional (elt)
  (if (atom elt)
    nil
    (eq (car elt) 'opt)))

(defun unstop (elt)
  (if (atom elt)
    nil
    (eq (car elt) 'unstop)))

(defun all-opt (phrase)
  (if (null phrase)
    t
    (if (or (optional (first phrase)) (unstop (first phrase)))
      (all-opt (rest phrase)))))

(defun first-pos (elt) (cadr elt))

(defun second-pos (elt) (caddr elt))
(defun expand (p grammar)
  (let ((index (hash p)))
    (if index
        (nth index grammar)
        (if (terminal p)
            (list p))))))

(defun hash (p)
  (cond ((eq p 'cp) 0)
            ((eq p 'ip-prime) 1)
            ((eq p 'ip) 2)
            ((eq p 'dp) 3)
            ((eq p 'vp-prime) 4)
            ((eq p 'np-prime) 5)
            ((eq p 'vp) 6)
            ((eq p 'np) 7)
            ((eq p 'v-prime) 8)
            ((eq p 'n-prime) 9)
            ((eq p 'advp) 10)
            ((eq p 'ap) 11)
            ((eq p 'pp) 12)
            ((eq p 'v+o) 13)))

(setq *the-terminals* '(N V Adj Adv D P Spec C Aux))

A.3 Miscellaneous

A.3.1 Search

(defun bf-search (parameters fit free)
  (defun bf-iter (param-s n)
    (if (= n *size-of-individual*)
      param-s
      (let ((imp (if (= free 3)
                    (try-change n param-s fit)
                    (try-s-change n param-s fit)))
            (if (= (calc_fitness imp) 1.0)
imp
  (bf-iter imp (1+ n))))
(let* ((new (bf-iter parameters 0))
       (nfit (calc_fitness new))
       (if (> 1.0 nfit)
           (bf-search new nfit)
           new)))

(defun try-change (n params fit)
  (let ((old (nth n params)))
    (cond ((= old 0)
            (let* ((params-1 (replacen n 1 params))
                   (params-2 (replacen n 2 params))
                   (fit-1 (calc_fitness params-1))
                   (fit-2 (calc_fitness params-2))
                   (max-fit (max fit fit-1 fit-2)))
                  (cond ((= max-fit fit) params)
                        ((= max-fit fit-1) params-1)
                        (t params-2)))
        ((= old 1)
         (let* ((params-0 (replacen n 0 params))
                 (params-2 (replacen n 2 params))
                 (fit-0 (calc_fitness params-0))
                 (fit-2 (calc_fitness params-2))
                 (max-fit (max fit fit-0 fit-2)))
          (cond ((= max-fit fit) params)
                ((= max-fit fit-0) params-0)
                (t params-2)))
        ((= old 2)
         (let* ((params-1 (replacen n 1 params))
                 (params-0 (replacen n 0 params))
                 (fit-1 (calc_fitness params-1))
                 (fit-0 (calc_fitness params-0))
                 (max-fit (max fit fit-1 fit-0)))
          (cond ((= max-fit fit) params)
                ((= max-fit fit-1) params-1))

37
(t params-0)))
(t (error "Invalid parameter: " old))))

(defun try-s-change (n params fit)
  (let ((old (nth n params)))
    (cond ((= old 0)
               (let* ((flip-params (replacen n 1 params))
                (flip-fit (calc_fitness flip-params)))
                (if (> flip-fit fit)
                    flip-params
                    params)))
      ((= old 1)
       (let* ((flip-params (replacen n 0 params))
                (flip-fit (calc_fitness flip-params)))
                (if (> flip-fit fit)
                    flip-params
                    params))))
  )))

A.3.2 Binary Tree

(defun lookup (word)
  (defun 1-iter (w tree)
    (if (null tree)
        nil
        (if (leafp tree)
            (if (string= w (string-downcase (first tree)))
                (rest tree)
                nil)
            (let ((top (root-head tree)))
                (if (string= w (string-downcase top))
                    (root-rest tree)
                    (if (string< w (string-downcase top))
                        (1-iter w (left-node tree))
                        (1-iter w (right-node tree)))))))
  (1-iter (string-downcase word) *lexical-db*))

38
(defun root-head (tree)
  (caar tree))

(defun root-rest (tree)
  (cdar tree))

(defun left-node (tree)
  (second tree))

(defun right-node (tree)
  (third tree))

(defun leafp (tree)
  (stringp (car tree)))

(defun create-btree (words)
  (let ((len (length words)))
    (cond ((= len 1) (list (first words) '() '()))
          ((= len 2) (list (first words) '() (second words)))
          ((= len 3) (list (second words) (first words) (third words)))
          (t
           (let* ((mid (truncate (* len 0.5)))
                   (sub1 (butlast words mid))
                   (sub2 (nthcdr (1+ mid) words)))
             (list (nth mid words)
                   (create-btree sub1)
                   (create-btree sub2)))))))
Appendix B

Input Sentences


excerpts from childes/eng/warren/george.cha:
I thought a leopard had spots.
Does the rabbit stamp on the ground?
How did that happen?
Did you make the heart?
What did you do with that cupcake?
You ate it.
What kind was it?
Did everybody like it?
Do you eat rocks?
Where do you eat rocks?
This is a water moccasin.
What kind of a mystery is this?

excerpts from childes/eng/warren/carol.cha:
You hit Janna in the nose?
Can you count?
I think it does.

excerpts from childes/eng-warren/beth.cha:

How do you use them at school?
You saw it on a commercial.
Why do they laugh at him?
Did you write that?

excerpts from childes/eng-warren/john.cha:

Have you got a launch?
They blast him?
You never heard of that one?
Were you working pretty fast?
What do you reckon?

excerpts from childes/eng-warren/susan.cha:

Who is Sassie?
A puppy is a baby dog.

These data were contributed by Patrick Suppes, Department of Philosophy, Stanford University (415) 723-3001. Suppes (1973) and MacWhinney and Snow (1985).

excerpts from childes/eng-suppes/nina34.cha:

what is poy doing?
he already had his supper.
what do dogs eat for supper?
what do you do with poy?
Mommy bought that for you.
what do little mice eat?
they eat cheese?
do we have a porch outside our house?
shall we put them on this?
you like red rabbits?
did you see other animals on the television?
would you make a whole building?

did you dance yesterday?

excerpts from childes/eng/suppes/nina35.cha:

did you find any eggs today?
do you like dried apricots?
did you keep it in the peel?
do we have a new kitty in our home?
she made a funny noise.
what are you putting in that pot?
I just took a big chunk of play dough.
what did you do at your new school yesterday?
could you make some grapes for me?
what does the silly bear do in the story?
do you think this is a pretty dress?

excerpts from childes/eng/suppes/nina38.cha:

that tastes very good.

he has very sharp prickers.

she has very big eyes.

excerpt from childes/eng/suppes/nina04.cha:

what is the animal that says cock+a+doodle-doo? -*

*-changed to read:

what is the animal that you saw that says cock+a+doodle-doo?

added (do not appear in database):

the pot that janna made hit the rocks that sassie saw.

the animals that janna saw thought she had big eyes.
Appendix C

Sample Results of Running evolve

> (evolve 100 2 50 0.9)
Generation 1: Best=(0 0 1 0 1 1 1 0 1 0 0 0 0), 39% fit
Generation 2: Best=(0 1 1 0 1 1 1 0 1 0 0 1 0), 57% fit
Generation 3: Best=(0 1 1 0 1 1 1 0 1 0 0 1 0), 57% fit
Generation 4: Best=(0 1 1 0 1 1 1 0 1 0 0 1 0), 57% fit
Generation 5: Best=(0 1 1 0 1 1 1 0 1 0 1 0 0), 88% fit
Generation 6: Best=(0 1 1 0 1 1 1 0 1 0 1 0 0), 88% fit
Generation 7: Best=(0 1 1 0 1 1 1 0 1 0 1 0 0), 88% fit
Generation 8: Best=(0 1 1 0 1 1 1 0 1 0 1 0 0), 88% fit
Generation 9: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
(0 1 1 0 1 2 2 2 2 2 2 0 1 1 0 0)
> (evolve 100 2 50 0.94)
Generation 1: Best=(0 1 1 0 1 1 0 1 0 0 1 0 0 0), 89% fit
Generation 2: Best=(0 1 1 0 1 1 0 1 0 0 1 0 0 0), 89% fit
Generation 3: Best=(0 1 1 0 1 1 0 1 0 0 1 0 0 0), 89% fit
Generation 4: Best=(0 1 1 0 1 1 0 1 0 0 1 0 0 0), 89% fit
Generation 5: Best=(0 1 1 0 1 1 0 1 0 0 1 0 0 0), 89% fit
Generation 6: Best=(0 1 1 0 1 1 0 1 0 0 1 0 0 0), 89% fit
Generation 7: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 8: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 9: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 10: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 11: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 12: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
<table>
<thead>
<tr>
<th>Generation</th>
<th>Best</th>
<th>fit</th>
</tr>
</thead>
<tbody>
<tr>
<td>Generation 13:</td>
<td>(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 14:</td>
<td>(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 15:</td>
<td>(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 16:</td>
<td>(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 17:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 18:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 19:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 20:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 21:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 22:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 23:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 24:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 25:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 26:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 27:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 28:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 29:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 30:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 31:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 32:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 33:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 34:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 35:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 36:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 37:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 38:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 39:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 40:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 41:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 42:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 43:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 44:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 45:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 46:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 47:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
<tr>
<td>Generation 48:</td>
<td>(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93%</td>
<td>fit</td>
</tr>
</tbody>
</table>
Generation 49: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 50: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 51: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 52: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 53: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 54: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 55: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 56: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 57: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 58: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 59: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 60: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 61: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 62: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 63: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 64: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 65: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 66: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 67: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 68: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 69: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 70: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 71: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 72: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 73: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 74: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 75: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 76: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 77: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 78: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 79: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 80: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 81: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 95% fit
(0 1 1 0 1 2 2 2 2 2 1 2 1 0 0)
> (evolve 50 2 100 0.94)
Generation 1: Best=(0 1 1 0 1 0 0 1 0 1 1 0 0 0), 82% fit
Generation 2: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 3: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 4: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 5: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 6: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 7: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 8: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 9: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 10: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 11: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 12: Best=(0 1 2 0 0 0 0 1 0 0 1 1 0 0), 95% fit
(0 1 2 0 0 0 2 1 0 0 1 1 0 0)
> (evolve 100 2 100)
Generation 1: Best=(1 1 0 0 1 1 1 1 0 0 0 1 1 1), 50% fit
Generation 2: Best=(0 1 1 0 1 0 0 1 1 0 0 1 0 0), 70% fit
Generation 3: Best=(0 1 1 0 1 0 1 1 0 1 1 1 0 0), 82% fit
Generation 4: Best=(0 1 1 0 1 0 1 1 0 1 1 1 0 0), 82% fit
Generation 5: Best=(0 1 1 0 1 0 1 1 0 1 0 1 0 0), 82% fit
Generation 6: Best=(0 1 1 0 1 0 0 1 0 0 1 0 0 0), 84% fit
Generation 7: Best=(0 1 1 0 1 0 0 1 0 0 1 0 0 0), 84% fit
Generation 8: Best=(0 1 1 0 0 1 1 1 0 1 1 1 0 0), 84% fit
Generation 9: Best=(0 1 1 0 1 1 1 1 0 1 1 1 0 0), 88% fit
Generation 10: Best=(0 1 1 0 1 1 1 1 0 1 1 1 0 0), 88% fit
Generation 11: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 12: Best=(0 1 1 0 1 1 1 1 0 1 1 1 0 0), 88% fit
Generation 13: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 14: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 15: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 16: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 17: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 18: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 19: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 20: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 21: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 22: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 23: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 24: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 25: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 26: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 27: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 28: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 89% fit
Generation 29: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 89% fit
Generation 30: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 89% fit
Generation 31: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 89% fit
Generation 32: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 33: Best=(0 1 1 0 2 0 0 1 0 1 1 1 0 0), 93% fit
Generation 34: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 35: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 36: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 37: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 38: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 39: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 40: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 41: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 42: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 43: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 44: Best=(0 1 1 0 0 0 1 1 0 1 0 1 0 0), 89% fit
Generation 45: Best=(0 1 1 0 0 0 1 1 0 1 0 1 0 0), 89% fit
Generation 46: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 47: Best=(0 1 1 0 1 1 1 1 0 0 1 1 0 0), 89% fit
Generation 48: Best=(0 1 1 0 1 1 1 1 0 0 1 1 0 0), 89% fit
Generation 49: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 50: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 51: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 52: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 53: Best=(0 1 1 0 1 2 0 1 0 1 1 1 0 0), 93% fit
Generation 54: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 55: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 56: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 57: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 58: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 59: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 60: Best=(0 1 1 0 2 1 0 1 0 0 1 1 0 0), 93% fit
Generation 61: Best=(0 1 1 0 2 1 0 1 0 0 1 1 0 0), 93% fit
Generation 62: Best=(0 1 1 0 2 1 0 1 0 1 1 1 0 0), 91% fit
Generation 63: Best=(0 1 1 0 2 0 0 1 0 1 1 1 0 0), 93% fit
Generation 64: Best=(0 1 1 0 2 0 0 1 0 1 1 1 0 0), 93% fit
Generation 65: Best=(0 1 1 0 2 0 0 1 0 1 1 1 0 0), 93% fit
Generation 66: Best=(0 1 1 0 2 0 0 1 0 1 1 1 0 0), 93% fit
Generation 67: Best=(0 1 1 0 2 0 0 1 0 1 1 1 0 0), 93% fit
Generation 68: Best=(0 1 1 0 2 0 0 1 0 1 1 1 0 0), 93% fit
Generation 69: Best=(0 1 1 0 2 0 0 1 0 1 1 1 0 0), 95% fit
Generation 70: Best=(0 1 1 0 2 0 0 1 0 1 1 1 0 0), 95% fit
Generation 71: Best=(0 1 1 0 2 0 0 1 0 1 1 1 0 0), 93% fit
Generation 72: Best=(2 1 1 0 1 2 0 1 0 1 1 1 0 0), 93% fit
Generation 73: Best=(0 1 1 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 74: Best=(0 1 1 0 0 0 1 1 0 0 1 1 0 0), 91% fit
Generation 75: Best=(0 1 1 0 0 0 1 1 0 0 1 1 0 0), 91% fit
Generation 76: Best=(0 1 1 0 0 0 1 1 0 0 1 1 0 0), 91% fit
Generation 77: Best=(0 1 1 0 0 0 1 1 0 0 1 1 0 0), 91% fit
Generation 78: Best=(0 1 1 0 2 1 0 1 0 1 1 1 0 0), 91% fit
Generation 79: Best=(0 1 1 0 2 1 0 1 0 1 1 1 0 0), 91% fit
Generation 80: Best=(0 1 1 0 1 2 1 1 0 0 1 1 0 0), 91% fit
Generation 81: Best=(0 1 1 0 1 2 1 1 0 0 1 1 0 0), 91% fit
Generation 82: Best=(0 1 1 0 1 2 1 1 0 0 1 1 0 0), 91% fit
Generation 83: Best=(0 1 1 0 1 2 1 1 0 0 1 1 0 0), 91% fit
Generation 84: Best=(0 1 1 0 2 0 1 1 0 0 0 1 0 0), 91% fit
Generation 85: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 86: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 87: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 88: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 89: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 90: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 91: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 92: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 93: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 94: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 95: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 96: Best=(0 1 1 0 1 1 0 1 1 1 0 0), 91% fit
Generation 97: Best=(0 1 1 0 1 1 0 1 0 1 1 0 0), 91% fit
Generation 98: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 99: Best=(0 1 1 0 0 0 1 1 0 0 0 2 0 0), 91% fit
Generation 100: Best=(0 1 1 0 0 0 1 1 0 0 0 1 0 0), 91% fit

(0.9107142857142857 (0 1 1 0 0 0 1 1 0 0 0 1 0 0))

> (evolve 50 2 100 1.0 2)

Generation 1: Best=(1 1 0 0 1 1 1 0 0 0 1 1 1), 53% fit
Generation 2: Best=(0 1 0 0 0 0 1 1 0 0 0 1), 58% fit
Generation 3: Best=(0 1 0 0 1 1 1 0 0 0 1 1 1), 58% fit
Generation 4: Best=(0 1 0 0 1 1 1 0 0 0 1 0 0), 60% fit
Generation 5: Best=(1 1 1 0 0 0 0 1 0 1 0 0 0 0), 75% fit
Generation 6: Best=(1 1 1 0 0 0 0 1 0 1 0 0 0 0), 75% fit
Generation 7: Best=(1 1 1 0 0 0 0 1 0 1 0 0 0 0), 75% fit
Generation 8: Best=(1 1 1 0 0 0 0 1 0 1 0 0 0 0), 75% fit
Generation 9: Best=(1 1 1 0 0 0 0 1 0 1 0 0 0 0), 75% fit
Generation 10: Best=(1 1 1 0 0 0 0 1 0 1 0 0 0 0), 76% fit
Generation 11: Best=(1 1 1 0 0 0 0 1 0 1 0 0 0 0), 75% fit
Generation 12: Best=(1 1 1 0 0 0 0 1 0 1 0 0 0 0), 75% fit
Generation 13: Best=(0 1 1 0 0 0 0 1 0 1 0 0 0 0), 91% fit
Generation 14: Best=(0 1 1 0 0 0 0 1 0 1 0 0 0 0), 91% fit
Generation 15: Best=(0 1 1 0 0 0 0 1 0 1 0 0 0 0), 91% fit
Generation 16: Best=(0 1 1 0 0 0 0 1 0 1 0 0 0 0), 91% fit
Generation 17: Best=(0 1 1 0 0 0 0 1 0 1 0 0 0 0), 91% fit
Generation 18: Best=(0 1 1 0 0 0 0 1 0 1 0 0 0 0), 91% fit
Generation 19: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 94% fit
Generation 20: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 94% fit
Generation 21: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0 0), 94% fit
Generation 22: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0 0), 94% fit
Generation 23: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0 0), 94% fit
Generation 24: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0 0), 94% fit
Generation 25: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 94% fit
Generation 26: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 94% fit
Generation 27: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0 0), 94% fit
Generation 28: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0 0), 94% fit
Generation 29: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0 0), 94% fit
Generation 30: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 31: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0), 98% fit
Generation 32: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0), 98% fit
Generation 33: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0), 98% fit
Generation 34: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 35: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 36: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 37: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 38: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0), 94% fit
Generation 39: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0), 94% fit
Generation 40: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 41: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 42: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 43: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 44: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 45: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 46: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 47: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 48: Best=(0 1 1 0 0 0 0 1 0 1 1 0 0), 94% fit
Generation 49: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0), 94% fit
Generation 50: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0), 94% fit
(0.9433962284150944 (0 1 1 0 0 0 0 1 0 1 0 1 0))
> (evolve 50 2 100 0.94 2)
Generation 1: Best=(0 1 1 0 0 1 1 1 0 0 1 1 0 0), 85% fit
Generation 2: Best=(0 1 1 0 0 1 1 1 0 0 1 1 0 0), 85% fit
Generation 3: Best=(0 1 1 0 0 1 1 1 0 0 1 1 0 0), 85% fit
Generation 4: Best=(0 1 1 0 0 1 1 1 0 0 1 1 0 0), 85% fit
Generation 5: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 100% fit
(0 1 1 0 0 0 0 1 0 1 1 0 0)
> (evolve 100 2 50 0.94)
Generation 1: Best=(0 0 1 0 1 1 1 1 0 1 0 0 0), 39% fit
Generation 2: Best=(0 1 1 0 1 1 1 1 0 1 0 0 0), 57% fit
Generation 3: Best=(0 1 1 0 1 1 1 1 0 1 0 0 1 0), 57% fit
Generation 4: Best=(0 1 1 0 1 1 1 1 0 1 0 0 1 0), 57% fit
Generation 5: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 6: Best=(0 1 1 0 1 1 1 0 1 0 1 0 0), 88% fit
Generation 7: Best=(0 1 1 0 1 1 1 0 1 0 1 0 0), 88% fit
Generation 8: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 9: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 10: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 11: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 12: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 13: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 14: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 15: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 16: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 17: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 18: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 19: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 20: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 21: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 22: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 23: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 24: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 25: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 26: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 27: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 28: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 29: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 30: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 31: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 32: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 33: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 34: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 35: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 36: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 37: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 38: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 39: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 40: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 41: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 42: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 43: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 44: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 45: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 46: Best=(0 1 1 0 1 1 2 1 0 1 0 1 0 0), 93% fit
Generation 47: Best=(0 1 1 0 1 1 2 1 0 2 0 1 0 0), 95% fit
(0 1 1 0 1 2 2 2 2 1 1 1 0 0)
> (evolve 50 1 100)
Generation 1: Best=(0 1 1 0 0 1 1 1 0 0 1 1 0 0), 86% fit
Generation 2: Best=(0 1 1 0 0 1 1 1 0 0 1 1 0 0), 86% fit
Generation 3: Best=(0 1 1 0 0 0 1 1 0 0 0 0 0 0), 88% fit
Generation 4: Best=(0 1 1 0 0 0 1 1 0 0 1 1 0 0), 91% fit
Generation 5: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 6: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 7: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 8: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 9: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 10: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 11: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 12: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 13: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 14: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 15: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 16: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 17: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 18: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 19: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 20: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 21: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 22: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 23: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 24: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 25: Best=(0 1 1 0 0 0 0 1 2 0 1 1 0 0), 95% fit
Generation 26: Best=(0 1 1 0 0 0 0 1 2 0 1 1 0 0), 95% fit
Generation 27: Best=(0 1 1 0 0 0 0 1 2 0 1 1 0 0), 95% fit
Generation 28: Best=(0 1 1 0 0 0 0 1 2 0 1 1 0 0), 95% fit
Generation 29: Best=(0 1 1 0 0 0 0 1 2 0 1 1 0 0), 95% fit
Generation 30: Best=(0 1 1 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 31: Best=(0 1 1 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 32: Best=(0 1 1 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 33: Best=(0 1 1 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 34: Best=(0 1 1 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 35: Best=(0 1 1 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 36: Best=(0 1 1 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 37: Best=(0 1 1 0 0 0 1 0 0 1 1 0 0), 95% fit
Generation 38: Best=(0 1 1 0 1 1 0 1 0 0 1 1 0 0), 93% fit
Generation 39: Best=(0 1 1 0 2 1 0 1 0 0 1 1 0 0), 93% fit
Generation 40: Best=(0 1 1 0 2 1 0 1 0 0 1 1 0 0), 93% fit
Generation 41: Best=(0 1 1 0 2 1 0 1 0 0 1 1 0 0), 93% fit
Generation 42: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 43: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 44: Best=(0 1 1 0 0 0 0 1 0 2 2 0 0), 95% fit
Generation 45: Best=(0 1 1 0 0 0 0 1 0 2 2 0 0), 95% fit
Generation 46: Best=(0 1 1 0 0 0 0 1 0 2 2 0 0), 95% fit
Generation 47: Best=(0 1 1 0 0 0 0 1 0 2 2 0 0), 95% fit
Generation 48: Best=(0 1 1 0 0 0 0 1 0 2 2 0 0), 95% fit
Generation 49: Best=(0 1 1 0 0 0 0 1 0 2 2 0 0), 95% fit
Generation 50: Best=(0 1 1 0 0 0 0 1 0 2 2 0 0), 95% fit

(0.9464285714285714 0 1 1 0 0 0 0 1 0 2 2 0 0))

> (evolve 50 1 100 0.94)
Generation 1: Best=(1 1 0 0 1 1 1 0 0 0 1 1 1), 50% fit
Generation 2: Best=(0 1 1 0 1 0 0 1 1 0 0 1 0 0), 70% fit
Generation 3: Best=(0 1 1 0 1 0 0 1 1 0 0 1 0 0), 70% fit
Generation 4: Best=(0 1 1 0 1 1 1 1 0 1 1 1 0 0), 88% fit
Generation 5: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 6: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 7: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 8: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 9: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 10: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 11: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 12: Best=(0 1 1 0 1 1 1 1 0 1 0 1 0 0), 88% fit
Generation 13: Best=(0 1 1 0 1 1 1 0 0 1 1 0 0), 89% fit
Generation 14: Best=(0 1 1 0 1 1 1 1 0 0 1 1 0 0), 89% fit
Generation 15: Best=(0 1 1 0 1 1 1 1 0 0 1 1 0 0), 89% fit
Generation 16: Best=(0 1 1 0 1 1 1 1 0 0 1 1 0 0), 89% fit
Generation 17: Best=(0 1 1 0 1 1 1 1 0 0 1 1 0 0), 89% fit
Generation 18: Best=(0 1 1 0 1 1 1 1 0 0 1 1 0 0), 89% fit
Generation 19: Best=(0 1 1 0 1 1 1 0 0 1 1 0 0), 89% fit
Generation 20: Best=(0 1 1 0 1 1 1 1 0 0 1 1 0 0), 89% fit
Generation 21: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 22: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 23: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 24: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 25: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 26: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 27: Best=(0 1 1 0 0 0 0 1 0 1 1 2 0 0), 93% fit
Generation 28: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 29: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 30: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 31: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 32: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 33: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 34: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 35: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 36: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 37: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 38: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 39: Best=(0 1 2 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 40: Best=(0 1 2 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 41: Best=(0 1 1 0 1 2 2 2 2 1 1 1 0 0), 96% fit
(0 1 1 0 1 2 2 2 2 1 1 1 0 0)
> (evolve 50 5 100)

Generation 1: Best=(1 1 0 0 1 1 1 1 0 0 0 1 1 1), 50% fit
Generation 2: Best=(0 1 1 0 1 0 0 1 1 0 0 1 0 0), 70% fit
Generation 3: Best=(0 1 1 0 1 0 1 1 0 1 1 1 0 0), 82% fit
Generation 4: Best=(0 1 1 0 1 0 1 1 0 1 1 1 0 0), 82% fit
Generation 5: Best=(0 1 1 0 1 0 1 1 0 1 1 1 0 0), 82% fit
Generation 6: Best=(0 1 1 0 1 0 1 1 0 1 1 1 0 0), 82% fit
Generation 7: Best=(0 1 1 0 1 0 0 1 0 1 1 1 0 0), 86% fit
Generation 8: Best=(0 1 1 0 1 0 0 1 0 1 1 1 0 0), 86% fit
Generation 9: Best=(0 1 1 0 1 0 0 1 0 1 1 1 0 0), 86% fit
Generation 10: Best=(0 1 1 0 1 0 0 1 0 1 1 1 0 0), 86% fit
Generation 11: Best=(0 1 1 0 1 1 1 0 1 1 1 0 0), 88% fit
Generation 12: Best=(0 1 1 0 1 1 1 0 1 1 1 0 0), 88% fit
Generation 13: Best=(0 1 1 0 1 1 1 0 1 1 1 0 0), 88% fit
Generation 14: Best=(0 1 1 0 1 1 1 0 1 1 1 0 0), 88% fit
Generation 15: Best=(0 1 1 0 1 1 1 0 1 1 1 0 0), 88% fit
Generation 16: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 17: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 18: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 19: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 20: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 21: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 22: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 23: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 24: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 25: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 26: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 27: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 28: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 29: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 30: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 31: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 32: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 33: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 34: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 35: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 36: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 37: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 38: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 39: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 95% fit
Generation 40: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 41: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 42: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 93% fit
Generation 43: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 44: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 45: Best=(0 1 1 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 46: Best=(0 1 1 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 47: Best=(0 1 1 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 48: Best=(0 1 1 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 49: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 50: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit

(0.9285714285714286 (0 1 1 0 0 0 0 1 0 1 1 1 0 0))
> (evolve 50 5 100 0.94)

Generation 1: Best=(1 1 0 0 0 0 1 0 0 0 1 1), 50% fit
Generation 2: Best=(0 1 1 0 1 0 0 1 1 0 0 0 1 0 0), 70% fit
Generation 3: Best=(0 1 1 0 1 0 1 1 0 1 1 1 0 0), 82% fit
Generation 4: Best=(0 1 1 0 1 0 1 1 0 1 1 1 0 0), 82% fit
Generation 5: Best=(0 1 1 0 1 0 1 1 0 1 1 1 0 0), 82% fit
Generation 6: Best=(0 1 1 0 1 0 1 1 0 1 1 1 0 0), 82% fit
Generation 7: Best=(0 1 1 0 1 0 0 1 0 1 1 1 0 0), 86% fit
Generation 8: Best=(0 1 1 0 1 0 0 1 0 1 1 1 0 0), 86% fit
Generation 9: Best=(0 1 1 0 1 0 0 1 0 1 1 1 0 0), 86% fit
Generation 10: Best=(0 1 1 0 1 0 0 1 0 1 1 1 0 0), 86% fit
Generation 11: Best=(0 1 1 0 1 1 1 0 1 1 1 0 0), 86% fit
Generation 12: Best=(0 1 1 0 1 1 1 0 1 1 1 0 0), 86% fit
Generation 13: Best=(0 1 1 0 1 1 1 0 1 1 1 0 0), 86% fit
Generation 14: Best=(0 1 1 0 1 1 1 0 1 1 1 0 0), 86% fit
Generation 15: Best=(0 1 1 0 1 1 1 0 1 1 1 0 0), 86% fit
Generation 16: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 17: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 18: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 19: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 20: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 21: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 22: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 23: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 24: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 25: Best=(0 1 1 0 0 0 1 1 0 1 1 1 0 0), 89% fit
Generation 26: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit

56
Generation 27: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 28: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 29: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 30: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 31: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 32: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 33: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 34: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 35: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 36: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 37: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 93% fit
Generation 38: Best=(0 1 1 0 1 1 0 1 0 1 1 1 0 0), 91% fit
Generation 39: Best=(0 1 1 0 0 0 0 1 0 1 1 1 0 0), 95% fit
Generation 40: Best=(0 1 1 0 0 0 2 1 0 0 1 1 0 0)

> (evolve 50 2 50 0.94)
Generation 1: Best=(0 1 1 0 0 1 0 1 0 1 0 1 0 0), 84% fit
Generation 2: Best=(0 1 1 0 0 1 0 1 0 1 0 1 0 0), 84% fit
Generation 3: Best=(0 1 1 0 0 1 0 1 0 1 1 1 0 0), 88% fit
Generation 4: Best=(0 1 1 0 0 1 0 1 0 1 1 1 0 0), 88% fit
Generation 5: Best=(0 1 1 0 0 1 0 1 0 1 1 1 0 0), 88% fit
Generation 6: Best=(0 1 1 0 0 1 0 1 0 1 1 1 0 0), 88% fit
Generation 7: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 89% fit
Generation 8: Best=(0 1 1 0 0 0 0 1 0 1 0 1 0 0), 89% fit
Generation 9: Best=(0 1 1 0 0 1 0 1 0 1 1 1 0 0), 88% fit
Generation 10: Best=(0 1 1 0 0 1 0 1 0 1 1 1 0 0), 88% fit
Generation 11: Best=(0 1 1 0 0 1 0 1 0 1 1 1 0 0), 88% fit
Generation 12: Best=(0 1 1 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 13: Best=(0 1 1 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 14: Best=(0 1 1 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 15: Best=(0 1 1 0 0 0 1 1 0 1 0 1 0 0), 89% fit
Generation 16: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 17: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 18: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 19: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 20: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 21: Best=(0 1 1 0 0 0 1 1 0 1 0 1 0 0), 89% fit
Generation 22: Best=(0 1 1 0 0 0 1 1 0 1 0 1 0 0), 89% fit
Generation 23: Best=(0 1 1 0 0 0 1 1 0 1 0 1 0 0), 89% fit
Generation 24: Best=(0 1 1 0 0 0 1 1 0 1 0 1 0 0), 89% fit
Generation 25: Best=(0 1 1 0 0 0 1 1 0 1 0 1 0 0), 89% fit
Generation 26: Best=(0 1 1 0 0 0 1 1 0 1 0 1 0 0), 89% fit
Generation 27: Best=(0 1 1 0 0 0 1 1 0 1 0 1 0 0), 89% fit
Generation 28: Best=(0 1 1 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 29: Best=(0 1 1 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 30: Best=(0 1 1 0 0 0 1 1 0 0 1 0 0), 91% fit
Generation 31: Best=(0 1 1 2 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 32: Best=(0 1 1 2 0 0 1 1 0 0 0 1 0 2), 91% fit
Generation 33: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 34: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 35: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 36: Best=(0 1 1 0 0 0 0 1 0 0 1 0 0), 91% fit
Generation 37: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 38: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 39: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 40: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 0), 91% fit
Generation 41: Best=(0 1 1 0 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 42: Best=(0 1 1 0 0 0 0 1 1 0 0 0 1 0 0), 91% fit
Generation 43: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 2), 91% fit
Generation 44: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 2), 91% fit
Generation 45: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 2), 91% fit
Generation 46: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 2), 91% fit
Generation 47: Best=(0 1 1 0 0 0 0 1 0 0 0 1 0 2), 91% fit
Generation 48: Best=(0 1 1 0 0 0 0 1 0 0 1 1 0 0), 95% fit
(0 1 1 0 0 0 2 1 0 0 1 1 0 0)
Bibliography

