The Lester and Sally Entin Faculty of Humanities

School of Philosophy, Linguistics and Science Studies

The Program of Cognitive Studies of Language and Its Uses

Learning morpho-phonology using the Minimum Description Length Principle and a Genetic Algorithm

MA thesis submitted by

Nur Lan

Thesis advisor:

Dr. Roni Katzir

May 2018

Abstract

We present an extension to Rasin et al.'s learner of morpho-phonological rules (2015), that was able to induce SPE-form representations from surface forms in an unsupervised way, by following the principle of Minimum Description Length (MDL) as an evaluation metric to select between possible grammars.

The original learner was hindered by computational limitations that restricted its application to limited corpora, both in size and resemblance to natural language. A genetic algorithm implementation is presented, replacing the previous learner's search procedure and allowing to speed up the search significantly. The improvement in performance allows the learner to tackle more complex corpora, resembling natural language.

The extended learner is presented with a corpus modeled after French opaque optional rule interaction, which was given by Dell (1981) as an example of a non-trivial learning challenge for the child learner. The learner is able to induce the underlying morphological and phonological representations of the French corpus.

Acknowledgments

I am, above all, grateful to my advisor Roni Katzir, whose guidance and insights, his breadth and depth of knowledge, have allowed this work, and the learning process that preceded it, to happen. This thesis is the outcome of a research held by Roni, in which I have been fortunate to take part. I could not have wished for a better path in my studies, which started in Roni's illuminating seminar about learning and culminated in fascinating research, to which I am proud to have contributed.

I am also thankful for having met Roni's team, and especially Ezer Rasin, who has shared his immense knowledge with endless patience, kindness and clarity, during many hours of work that have enriched me academically, professionally and personally.

Without Iddo Berger, who wrote the learner's code and implemented its algorithms, this project would not have happened. Luckily for me it is one of the most elegant and clean pieces of code I have seen. It is thanks to Iddo's limitless talent, guidance and generosity that I was able to contribute to this project.

I would also like to thank my teachers in the cognitive aspects of language program, in the Sagol school of neuroscience and in the linguistics department at

Tel Aviv University: Orna Peleg, Yair Lakretz, Outi Bat-El, and Irena Botwinik. I also thank my friends and fellow students Aviaz Rand, Dror Chawin, and Victoria Costa.

Finally I would like to thank my family and friends, and my love Adi Chawin for bearing my endless staring at dark screens filled with simulation logs. I would especially like to thank my cat friend Puti (Ruti) for her endless love and purrs.

Contents

1 MDL Learning **9**

1.1 Introduction 9

1.1.1 The MDL criterion 14

1.2 Learning morpho-phonology 15

1.2.1 Hypothesis representation 15

1.2.1.1 Phonological rules 16

1.2.1.2 Lexicon 18

1.2.1.3 Data given the grammar 19

1.2.2 Search 21

2 Genetic Algorithm **25**

2.1 Background 25

2.2 Basic Genetic Algorithm 26

2.3 Application 30

2.4 Morphophonology learning 31

2.4.1 Population initialization 32

2.4.1.1 Random lexicon 33
2.4.1.2 Random rules 33
2.4.2 Mutation 34
2.4.3 Crossover 34
2.4.4 Lexicon Crossover 34
2.4.4.1 Transition matrix crossover 35
2.4.4.2 Connected components crossover 37
2.4.4.3 Subgraph crossover 39
2.4.4.4 Unilateral emissions crossover 41
2.4.5 Rule Crossover 42
2.4.5.1 Rule pair crossover 42
2.4.5.2 Unilateral rule crossover 43
2.4.6 Selection 44
2.4.6.1 Rank-based selection 44
2.4.6.2 Tournament selection 46
2.4.7 Fitness 47
2.4.8 Elitism 49
2.4.9 Parallelization 49
2.4.9.1 Naive parallelization 50
2.4.9.2 Island model 51
2.4.10 Technical information 56
2.4.11 Performance 56
2.4.12 Hyperparameters comparison 57

2.4.12.1 Crossover and mutation rates 57
2.4.12.2 Island population size 60
2.4.12.3 Elite size 60
2.4.12.4 Crossover operators 63
2.4.12.5 Selection methods 63

3 French rule interaction **65**
3.1 Background 65
3.1.1 Rule interaction 67
3.2 Simulation 70
3.2.1 Results 71

4 Discussion **75**

A Mutations list **77**
A.1 Mutations on HMM 77
A.2 Mutations on feature bundle list 79
A.3 Mutations on rule set 79

B French simulation data **81**

Chapter 1

MDL Learning

1.1 Introduction

Facing a complex reality, the cognitive system copes surprisingly well with the vast amounts of data it perceives. One long-lasting explanation for how humans learn under these conditions claims that the cognitive system inherently strives for simplicity. That is, that whenever two competing hypotheses try to explain certain perceived phenomena, the cognitive system would prefer the simplest.

Over the years, the claim for a simplicity criterion has had numerous incarnations in various fields. It can be traced back to 'Occam's Razor', a principle attributed to 14th century's William of Occam which states that *entities should not be multiplied beyond necessity*. This is usually interpreted like so: when competing theories are compared, and all are consistent with the observed data, the theory that provides the simplest explanation of the data should be preferred. Jumping

ahead to the 19th and 20th century, a similar reasoning can be found in the Gestalt theory, where a simplicity criterion was used to explain perceptual phenomena - for example, that the visual system automatically completes the outline of occluded shapes (Chater, 1999).

Several decades later, the elusive idea of simplicity has found a quantifiable interpretation in the form of Kolmogorov Complexity, a mathematical and information-theory incarnation of Occam's Razor. Building upon the pioneering work by Solomonoff (1964) and developed independently by Kolmogorov (1965) and Chaitin (1966), Kolmogorov complexity parallels simplicity with the tangible definition of *description length*. Instead of isomorphically identifying an object with its content, Kolmogorov complexity considers the object's description length - or more precisely, the length of a Turing machine or equivalent program that prints the object and halts - as a measure of its simplicity.

For example, say we are given two infinite sequences of 1's and 0's, and are asked to decide which one is 'simpler':

$$0101010101010101010101010101010101... \tag{1.1}$$

$$0011101000110101110000010101110101... \tag{1.2}$$

(1.1) would be universally selected as the simpler sequence. But why? Using Kolmogorov complexity we can quantify this intuition: (1.1) can be described in a much more compact way than (1.2). Say we choose our description to be

nondeterministic finite automata (NFA), all we need is this simple representation:

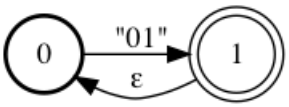


Figure 1.1: NFA generating string (1.1)

However, in order to describe (1.2), we would need an automaton of at least the length of (1.2) itself. This is because (1.2) contains no regularities that can be used to describe it in a way that would be more concise than its surface form. This notion can also be used to define another elusive idea: randomness; an object may be considered random if no regularities can be found to compress it.

Using the notion of description length, we can see how a quest for simplicity, now a quantifiable term, can yield learning: in order to find the simplest explanation of an object - that is, perceived data or phenomena - we look for its shortest description, e.g., the shortest program that will produce it. In order to do that, we need to find regularities in the object, which will help us compress it.

Using these terms, learning can be seen as finding regularities in an object and compressing it in order to minimize its description. Applying the learned program can be seen as generalizing beyond the data. In our example, the regularity ”0 and 1 repeated forever” was learned and used to compress the sequence to a shorter program. Re-running the program, to produce a continuation of the string, can be seen as generalizing beyond the data.

A common risk in learning is that of *overgeneralization*. Using description length as a guiding factor for learning helps to overcome this problem: by def-

inition, the Kolmogorov complexity of an object is derived from the description of the object itself, and only it (i.e. the Turing machine that prints that object and halts). By restricting the description to the object itself, generalization is also restricted and overgeneralization is prevented. For example, for the binary string example above, a program or a NFA that generates all possible strings that contain 1's and 0's would still describe (1.1) and will also be shorter than the NFA in 1.1; however, it will not qualify as the minimum description since it generates more objects than the required one, and overgeneralization is prevented.

Language acquisition is a learning process that has drawn arguments for a simplicity criterion (Chater and Vitányi, 2007). The language learning child acquires language within a relatively short time (Chomsky, 1965), during which they need to select between an infinite number of possible grammars that fit the utterances they face. It is widely assumed that the child is met with mostly positive examples (Marcus, 1993), which poses an overgeneralizaiton problem - a learner that is only exposed to positive data might quickly start to overgeneralize; this calls for a learning mechanism that prevents overgeneralization.

Numerous attempts have been made at applying a simplicity criterion to language acquisition. This work extends that of Rasin et al. (2015)[1] which deals with the specific task of learning rule-based morpho-phonology, and that of Rasin and Katzir (2016) which tries to acquire linguistic knowledge in the same domain but

[1]The existing rule-based learner by Rasin et al. is referenced throughout this work as Rasin et al. (2015) for an earlier version that used simulated annealing as its search procedure, described in section 1.2.2; and as Rasin et al. (2018) for a version which already uses the genetic algorithm described in this work. Other than that the works are interchangeable.

using constraints-based phonology.

Already in Chomsky and Halle's Sound Pattern of English (SPE; 1968, p.334) an evaluation metric was given for comparing phonological grammars, based on what can be interpreted as a simplicity principle: given two competing grammars G and G' that can both generate the data, and $|G| < |G'|$ - G should be preferred. $|\cdot|$ represents the number of symbols in each grammar.

Similarly to the Kolmogorov example above, it can be seen how using this economy metric yields generalization. However, it can also be seen that a child learner using this metric will quickly begin to overgeneralize, since the metric poses no limit on the minimization of the grammar as long as it describes the data. Moreover, it does not take into account rule ordering, since is does not provide a mechanism for discriminating between two hypotheses as long as they have equal number of symbols, albeit with different rule orderings. These two issues will become significant in the French example given in Chapter 3.

Since the SPE economy metric is missing a component to balance overgeneralization, a natural complementary evaluation metric would allow to restrict grammars to the data observed. This is taken to the extreme in the subset principle (Berwick, 1985; Wexler and Manzini, 1987), which states the following: in choosing between two competing grammars G and G', which describe the data equally well and generate two sets of licit surface forms L and L', and $L \subsetneq L'$ (L is a strict subset of L') - prefer G over G'.

With this metric, the problem of overgeneralization is solved: the learner will always prefer the grammar that is most restrictive with respect to the data. How-

ever, it can be seen how this leads to a problem at the other extreme - that of undergeneralization ('overfitting'). A learner following the subset principle will fail to generalize beyond the data, because a generalizing grammar will always generate a language that is a superset of the language generated by a more restricted grammar, that generates only the data itself.

1.1.1 The MDL criterion

It is thus clear that an evaluation metric that will allow to generalize beyond the data, while restricting the grammar to the perceived data, is needed in order to allow proper learning. As seen in the previous sections, Kolmogorov complexity embodies both these qualities.

While Kolmogorov complexity is not computable (Li and Vitányi, 2008), it might still be computable for specific problem encodings that limit the hypothesis space. We will use an approximation to Kolmogorov complexity, the Minimum Description Length principle (MDL; Rissanen, 1978). For our context of evaluating competing grammars, the principle postulates the following; when choosing between grammars, given the observed data D , choose the grammar G that minimizes the sum:

$$|G| + |D{:}G| \tag{1.3}$$

In the context of our learner of morpho-phonology grammars, the two summands in (1.3) have the following meaning:

$$\underbrace{\underbrace{0110100100100000011011000110}_{Lexicon}\underbrace{10101010010}_{Rules}}_{G}\underbrace{01100001011001000110100 1}_{D:G}$$

- $|G|$ - the number of bits used to encode the grammar G, which consists of a lexicon and a rule set.
- $|D{:}G|$ - the number of bits used to encode the data D given that G was internalized.

By minimizing the two factors in (1.3), the learner balances between an over-restrictive grammar and an over-generalizing one. This allows to generalize while avoiding overgeneralization, and to encode exceptions without overfitting the data. If G gets too general, $|G|$ will become smaller, while $|D{:}G|$ will grow in order to encode exceptions. Inversely, if the grammar memorizes the data instead of generalizing, $|D{:}G|$ will diminish because all exceptions will be encoded in G, but $|G|$ will grow significantly.

1.2 Learning morpho-phonology

1.2.1 Hypothesis representation

In order to apply the MDL principle to learning rule based morpho-phonology, it is needed to measure the encoding length of the hypotheses that make up the

learner's hypothesis space. For this it is needed to work out the representation and encoding scheme for morpho-phonological hypotheses. This section explains the learner's inner workings and mostly summarizes the work done in Rasin et al. (2015).

As is customary in phonology, phonetic segments are not represented atomically but as feature bundles. The segments and features are aligned in a table like the one in Figure 1.2. For our learner, due to performance limitations and to facilitate the search, current simulations have specific feature tables that contain only the necessary segments for the corpus.

	cons	*voice*	*velar*	*cont*	*back*
d	+	+	-	-	-
t	+	-	-	-	-
g	+	+	+	-	-
k	+	-	+	-	-
z	+	+	-	+	-
s	+	-	-	+	-
i	-	+	-	+	-
u	-	+	-	+	+

Figure 1.2: Example feature table

1.2.1.1 Phonological rules

Rules are represented in standard SPE format, as shown in Figure 1.3. A and B are feature bundles like the one in Figure 1.4 and may be empty ($\emptyset$). X and Y are sequences of feature bundles that may also be empty, and constitute the left and right contexts of the rule application environment. *optional* is a boolean flag specifying whether the rule is optional or obligatory.

$$\underbrace{A}_{\text{focus}} \rightarrow \underbrace{B}_{\text{change}} / \underbrace{X}_{\text{left context}} \_\_ \underbrace{Y}_{\text{right context}} \text{ (optional?)}$$

Figure 1.3: Rule format

$$\begin{bmatrix} +cons \\ -voice \end{bmatrix}$$

Figure 1.4: Example feature bundle

In order to convert rules to binary representation, we can first convert them to intermediary representation as strings, that will later be converted to bit strings. Below is an optional liquid-deletion rule, given in (1a) in standard SPE notation and in (1b) in its intermediary flat string notation. The string contains all the components of the rule, separated by delimiters (#) that are needed to reconstruct the original rule from its flattened string and binary forms.

(1) Liquid deletion rule

a. SPE notation

$$\begin{bmatrix} +liquid \end{bmatrix} \rightarrow \emptyset / \begin{bmatrix} +cons \\ -son \end{bmatrix} \_\_ \text{ (optional)}$$

b. String notation

$$+liquid\#_{rc}\emptyset\#_{rc} + cons - son\#_{b}\#_{f}1\#_{rc}$$

In order to convert the intermediary string to a binary string, a conversion

table can be used that converts all available symbols to binary strings. The length of each symbol is determined by the number of all possible symbols in a rule (plus the delimiters). For this, a naive binary encoding is used, i.e. if there are n possible symbols, each binary string will be of length $\lceil \log_2 n \rceil$ bits.

A phonological grammar can contain multiple rules, and the order of their application is significant; each rule is thus converted to its binary string representation, and the strings are concatenated in order.

1.2.1.2 Lexicon

The learner will jointly learn morphological and phonological representations, so it is also needed to represent the lexicon of underlying representations (URs). This is done using Hidden Markov Models (HMMs), whose emissions are used to store morphemes, and state transitions are used to represent possible morpheme combinations. An example HMM for the URs /kat/ and /dog/ and the optional plural suffix /z/ is given in Figure 1.5.

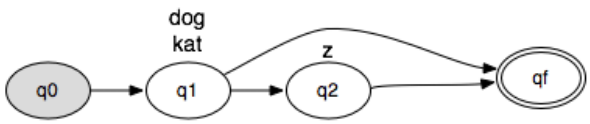


Figure 1.5: Plural English lexicon represented by an HMM

In order to encode the lexicon HMM and to be able to measure its length, the HMM is serialized into a binary string using a conversion table similar to the one constructed for the rule set. Each symbol in the final encoding is a binary string of

length $\lceil \log_2 n \rceil$ bits, where n is the number of elements required to represent an HMM: all states, segments (that make up emissions), and delimiters that enable to reconstruct the HMM. Each symbol in this string will be converted to a binary string, and these will be concatenated into the final binary representation.

1.2.1.3 Data given the grammar

In order to measure the encoding length of the $|D{:}G|$ term in the MDL sum, it is needed to compile the lexicon and the rules in a way that will express encoding choices of the surface forms in D given G. For example, application of obligatory rules would not incur an encoding length cost since they always apply, whereas selecting a morpheme among a HMM state's possible emissions, or application of optional rules, would incur a cost proportional to the number of choices to be made along the way.

Generating all the possible outputs of a grammar by brute-force is unfeasible (e.g. the grammar may generate infinite outputs), hence a more efficient parsing method is needed. Parsing is thus done by transforming both lexicon and rules into finite-state transducers (FSTs), and then composing them onto one another to receive the final grammar FST. The compilation of SPE rewrite rules into FSTs is done using the method described in Kaplan and Kay (1994) and was implemented for the learner in Rasin et al. (2015). The lexicon HMM is transformed into a FST by flattening the HMM's state transitions and emissions into a NFA (later converted to FST), in which state transitions become epsilon transitions and emissions are written on the FST's input and output arcs.

The outputs of the final grammar FST represent all surface forms generated by the grammar (a grammar may generate more or less outputs than the correct grammar). In order to calculate $|D{:}G|$ for a single surface form, the FST is parsed for all possible derivations of it, and the encoding length is set to the number of bits required to encode each choice. A naive binary encoding is used, in which a selection between n possibilities has a uniform probability and costs $\lceil \log_2 n \rceil$ bits[2].

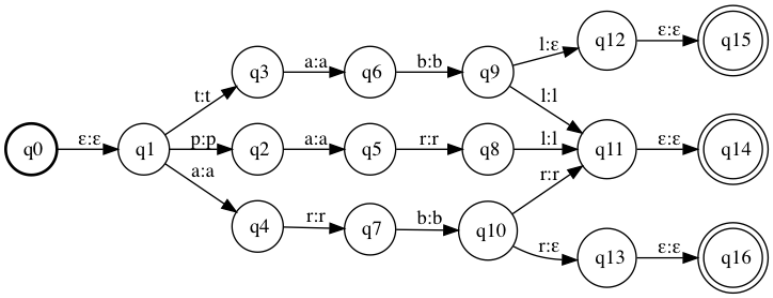


Figure 1.6: FST for a grammar with a liquid deletion rule and a lexicon containing /tabl/, /arbr/ and /parl/.

An example FST for a grammar consisting of the liquid-deletion rule (1a) and a lexicon with three underlying forms /tabl/, /arbr/, and /parl/ is given in Figure 1.6.

Let us follow a sample calculation of the encoding length of the surface form [tab] - the result of optionally deleting the liquid at the end of /tabl/. First, the

[2]The use of a uniform, non-entropy encoding is done for the sake of simplicity and explainability of the model. It is however a wasteful encoding scheme, e.g. if a choice is made among 9 possibilities, it would be encoded using 4 bits, and 7 binary strings will be left unused. This makes the model highly sensitive to variations in the grammar transducer, where a single extra state that happens to push the number of choices beyond a power of 2 can cause the final encoding length to grow significantly. This behavior may introduce unwanted artifacts to the final grammar, see discussion in Chapter 3 and in Rasin et al. (2018).

transition from q_0 to q_1 is deterministic and costs 0 bits. Then, any outgoing transition from q_1 costs $\lceil \log_2 3 \rceil = 2$ bits, representing the selection of one of the three possible morphemes. We take the transition to q_3 representing /tabl/. The next non-deterministic choice is specifying the transition from q_9 to either q_{12} or q_{11}, which represents specifying whether /l/ should be deleted and costs $\lceil \log_2 2 \rceil = 1$ bit. The next transition to the final accepting state is deterministic and costs 0 bits. The final encoding length for [tab] is thus 3 bits. This process is repeated for each surface form in the corpus and summed to form $|D{:}G|$. It can also be seen that for /parl/ the encoding length would be only 1 bit for the morpheme selection, since the deletion rule does not apply in the environment.

1.2.2 Search

The encoding scheme described above makes it possible to assign a description length $|G| + |D{:}G|$ to a hypothesis which consists of a lexicon and rule set. The goal of the learner is to find the hypothesis with the minimal description length, and for this a search procedure is needed. Since the hypothesis space is infinite, brute-force search for the optimal hypothesis is infeasible. Moreover, since the hypotheses that the learner is manipulating are discrete objects which don't convert to continuous representations, and because the MDL target function is not differentiable, the task is irrelevant to gradient-based optimizers.

Simulated Annealing (SA; Kirkpatrick et al., 1983) was used in Rasin et al. (2015) as the optimization method for the learner. SA is inspired by a physical technique of slowly cooling down a heated metal, with short periodic heating

phases, in order to minimize the energy of the material.

One of the advantages of SA is that it can operate on discrete objects like the learner's hypotheses, and that it is designed to reach a global optimum in complicated hypothesis spaces. Similarly to its physical inspiration, SA has two main configurable hyperparameters: an initial *temperature*, and a constant fraction called the *cooling rate*. The algorithm starts with some initial hypothesis that will be described below. At each step of the algorithm's run, the temperature T is lowered by multiplying it with the cooling rate. The algorithm ends when T reaches a bottom configurable threshold. Inspired by the equivalent physical process, the target function that assign scores to hypotheses is called the *energy* function, E. For MDL learning the energy function value is $|G| + |D{:}G|$, and since we're minimizing, lower energy is better.

At each step of the algorithm, the hypothesis from the previous step undergoes a mutation that generates a 'neighbor' hypothesis by changing either the rule set or the lexicon. The full list of mutations is given in Appendix A. If the neighbor has a better energy, it is always switched to. If not, the algorithm stochastically chooses to switch according to a probability that is dependent on the number of steps it has completed so far (expressed through T) and the difference in energy between the two hypotheses. The probability of switching is given in (1.4).

$$P_{switch} = e^{\frac{-(E_{neighbor} - E_{current})}{T}} \tag{1.4}$$

In the beginning of the algorithm, when the temperature is relatively high, the

decision will be less greedy and the search will tend to make large leaps across the search space by switching hypotheses, even for worse. As the search progresses and temperature is lowered, the algorithm will become more greedy and will prefer local improvements of the current hypothesis, that will hopefully lead it to the global optimum. The hypothesis at the algorithm's final step is declared as the optimal one found.

For MDL learning, the initial hypothesis was set to a naive grammar that contains no phonological rules and a lexicon that generates $\Sigma*$. This hypothesis is far from the MDL-optimum since it overgeneralizes and incurs an expensive $|D{:}G|$. Another implementation choice made for MDL learning was to only assign energy values to valid hypotheses, i.e. if mutation yields a hypothesis that fails to represent the data, it is never switched to.

While SA has worked well in Rasin et al. (2015) and successfully found the optimal hypotheses for some datasets, its main disadvantage lies in its very long run time: even for small corpora that consisted of 32 surface forms, simulation run times were in the days and even weeks[3]. First, this is because long run times are an inherent part of the algorithm - the slower the cooling rate is, the better the chances are of finding an optimal solution. Secondly, SA is a sequential algorithm that manipulates only one hypothesis at a time, and so even strong multiprocessor machines cannot be used to improve its performance (attempts at parallelization are discussed in Section 2.4.9). In order to support larger corpora with more complex phonological phenomena that resemble natural language, the search proce-

[3]These simulations were run on Intel Xeon 3.30GHz machines with 16GB RAM.

dure needed to improved.

The next chapter will present an alternative search procedure for the learner, replacing simulated annealing with a genetic algorithm that improves performance significantly and enables testing the learner on larger and more realistic corpora.

Chapter 2

Genetic Algorithm

2.1 Background

Since the early days of the computer age and artificial intelligence research, researchers have been using the growing knowledge about natural biological systems in order to model them computationally, to better understand these systems and to utilize them in novel ways.

In the field of neuroanatomy, for example, discoveries that spanned over centuries about how neurons are structured and how they operate have led to the development of models known today as artificial neural networks - a mathematical abstraction of complex cerebral structures. Later on, revelations about the visual system led to development of more advanced neural networks that mimic the working principles of the visual cortex. Similarly, the theory of Darwinian biological evolution has inspired researchers to try and model its mechanism. This

has led to a domain known today as Evolutionary Computing, which includes a wide range of algorithms and methods inspired by evolutionary principles, of which genetic algorithms (GAs) are the most commonly used.

Genetic algorithms were developed by John Holland in the 1960s, first as a straightforward approach to model and understand evolutionary phenomena, and then in Holland (1975) as an optimization method for computational problems (Mitchell, 1998a). Genetic algorithms arise from the notion that natural evolution can be thought of as an 'optimization' of a species, which is improved ad infinitum by the basic mechanisms of evolution: natural selection, mating, and mutation. Genetic algorithms abstract these mechanisms, and use them to help solve computational problems. Typically these problems will have no known analytical solution, and their set of possible solutions will be too vast to traverse using brute-force search.

2.2 Basic Genetic Algorithm

The canonical genetic algorithm represents each possible solution to the problem in a binary encoding, as a string of bits. Each such string is called a *chromosome*, a bit in the string/chromosome is called a *gene*, and each of its possible values (0 or 1) is called an *allele*. The terms are naturally borrowed from biology although they have more abstract roles in the algorithm. The basic genetic algorithm is presented in Algorithm 1.

When the algorithm is first started, an initial set of chromosomes are gener-

Algorithm 1 Basic Genetic Algorithm

$F \leftarrow$ fitness function
$S \leftarrow$ selection function
$X \leftarrow$ crossover operator
$M \leftarrow$ mutation operator
$N \leftarrow$ population size
$p_x \leftarrow$ crossover rate
$p_m \leftarrow$ mutation rate

initialize population P with N random hypotheses
evaluate fitness for each $h_i \in P$ using F

$generation \leftarrow 0$
while $generation <$ *max-generations* **do**:
    for each pair $h_i, h_{i+1} \in P$, apply crossover X with probability p_x to produce two offspring, h'_i, h'_{i+1}
    $h_i, h_{i+1} \leftarrow h'_i, h'_{i+1}$
    for each $h_i \in P$ apply mutation M with probability p_m to produce h'_i
    $h_i \leftarrow h'_i$
    evaluate fitness for each $h_i \in P$ using F
    $P \leftarrow S(P)$
    $generation \leftarrow generation + 1$
end while

return argmax$_{h \in P} F(h)$

ated randomly. This is called the *population*, and each member of it considered an *individual*. Each chromosome is assigned a *fitness* score, which represents its quality in the context of the problem being solved. For maximization problems, a larger fitness score is better, and vice versa for minimization. Throughout the algorithm's run, the population undergoes three basic processes inspired by biological evolution: *crossover*, *mutation*, and *selection*. Each step of crossover-mutation-selection is called a *generation*, and the algorithm is stopped after a predefined number of generations, or when the population reaches a predefined stopping criterion. The fittest chromosome in the population is then taken as the optimal solution.

Crossover, sometimes called *mating*, consists of selecting pairs of individuals from the population and combining them to produce two new offspring. This combination needs to be implemented in a meaningful way so that each offspring will inherit useful characteristics from its parents. Thus, two well-performing parents will hopefully reproduce to create offspring who will inherit their parents' features. A configurable probability called the *crossover rate* is used to choose whether a pair of parents undergo crossover.

The *mutation* operator is then applied to each offspring, according to a probability called the *mutation rate*. Inspired by genetic mutation that occurs in biological organisms due to environmental factors, the mutation operator randomly changes one or more genes in the chromosome. In the basic binary-representation genetic algorithm, a mutation consists of flipping one or more bits of the chromosome. Similarly to in nature, more often than not a random mutation will harm an

individual's fitness. Every once in a while, though, a mutation may prove advantageous and equip the individual with a feature that will increase its fitness.

The *selection* phase, inspired by natural selection ("survival of the fittest"), is then applied to select the individuals that will survive onto the next generation. The selection operator assigns a survival probability to an individual, usually as an increasing function of its fitness combined with a random chance of survival. The degree to which the selection operator favors better-performing individuals is called the *selection pressure*. A high selection pressure will push the algorithm towards a more greedy behavior, in which it might ignore individuals that could have developed into good solutions, and increases the chances of getting stuck in a local optimum. A selection pressure too low, on the other hand, might slow the algorithm down to the level of random search.

Another important effect of selection pressure lies in its interaction with the mutation operator: the advantage gained by features acquired through mutation may not be immediately visible; a chromosome may need to carry an allele reached through mutation for several generations until it can be used. The selection pressure thus needs to be lenient enough so as to allow less-performing genes to remain dormant for a while[1].

[1]For example, Tattersall (2012) notes how homo-sapiens may have acquired genetic and cognitive traits many generations before they turned out useful, for uses like erect walk, toolmaking, symbolism and language.

2.3 Application

A successful optimization method is often said to balance between two principles: *exploration* and *exploitation*. Exploration describes the global search that is needed in order to scan a satisfactory portion of the solution space. Exploitation describes the local search that occurs when the algorithm opts for a good enough solution, and tries to improve on it. Genetic algorithms satisfy both requirements: the genetic crossover operator can be seen as exploitation of good characteristics found in parents and distilled onto their offspring; and the mutation operator can be seen as exploration, using randomness to tread into unknown areas of the solution space. Good exploratory power is essential when dealing with infinite hypothesis spaces, like our learner's.

Another advantage of genetic algorithms is that they do not require the problem's solution space to be reduced to a continuous representation, i.e. a numerical vector. The basic genetic operators - crossover and mutation - can be implemented to apply to symbolic representations, as we will see in the upcoming sections. This means that genetic algorithms can operate on discrete symbolic objects, while other models and optimization methods - like neural networks and other gradient descent optimizers - require the problem to be put in numerical terms; they also require the target function to be differentiable, while MDL isn't. This makes genetic algorithms a good candidate for traversing a solution space like the one faced by our learner, that is comprised of discrete lexicons and rules.

Yet another advantage of genetic algorithms is their intrinsic parallelizability.

In each step of the algorithm, the three basic genetic operators - crossover, mutation, and fitness calculation - are applied to each individual in the population (or to pairs of chromosomes, in the case of crossover). Since each application is independent of the others' results, this process can be easily parallelized, resulting in a potential speedup linear in the number of parallel processes. These advantages make genetic algorithms a good candidate for our learner's search procedure.

At the same time, it should be noted that the search algorithm is chosen here solely for its performance and ability to optimize the MDL metric for morphophonological hypotheses as they are formalized in this work. We assume nothing about the way the child learner's brain actually performs the search, of which we still know very little.

2.4 Morphophonology learning

The following genetic algorithm implementation replaces the simulated annealing optimization described in Section 1.2.2 as the search method for the learner, in order to improve its performance.

The representation of hypotheses is kept the same, using HMMs to represent lexicons, and sets of SPE rules. Working in the context of SPE rules and morphological lexicons, the basic genetic operators needed to be implemented in a way that will be expressive yet meaningful enough so as to allow the algorithm to learn effectively.

Another important aspect to bear in mind for this implementation is that the

search method plays a different role for our learner than it usually does in optimization challenges. Our goal was to demonstrate how MDL - or more broadly, simplicity - can operate as an evaluation metric for the child learner. This puts our learner in a different context than other problem-solving tasks: unlike machine learning tasks in an engineering context, that aim to find the best solution quickly and may use as much supervision as possible - our learner's goal is to find an optimal solution while being as unsupervised and agnostic about the search landscape as possible; this, in order to demonstrate how simplicity can act as a guiding factor alone. Problem-specific tweaks and tricks, like limiting the learner to specific mutations that we know are useful, could have been put in place to help guide the search towards the wanted solution. Instead, the learner is willingly left with an agnostic (yet expressive) toolbox, and is able to induce correct hypotheses.

Sections 2.4.1-2.4.10 describe the implementation in detail. Section 2.4.9 describes parallelization techniques for increased performance. Section 2.4.11 discusses the increase in performance. Section 2.4.12 presents an empirical comparison of different parameters of the algorithm.

2.4.1 Population initialization

Upon simulation start, a random population of size N is generated randomly. As with the earlier simulated annealing-based learner, each hypothesis in the population consists of a HMM representing the lexicon, and a rules set.

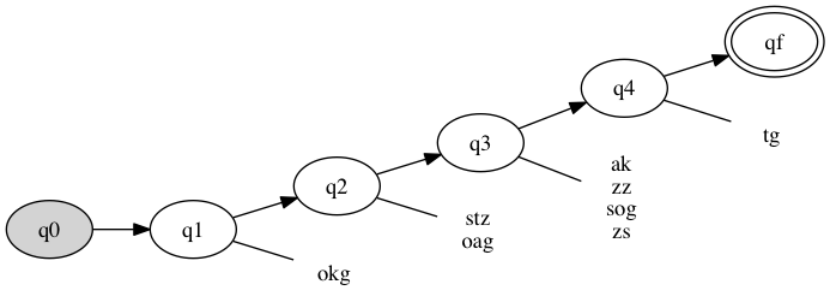


Figure 2.1: Random HMM example

2.4.1.1 Random lexicon

A random HMM representing the lexicon is created by generating a chain of states, with a random number of states leading from the initial state q_0 to the final state q_f. Each state is assigned random emissions, using the table of available segments. The maximum number of emissions per state and their maximum length are configurable parameters of the simulation. An example random HMM is given in Figure 2.1.

2.4.1.2 Random rules

A random number of rules are generated, each one with a random feature bundle for each of the target, change and context positions. In case of simulations with word or morpheme boundary enabled, these flags are also randomized (with true/-false values). The maximum number of rules in a set and the maximum number of features per position are configurable parameters of the simulation. An example random rules set is given in Figure 2.2.

$$R_1 : [+cons] \rightarrow [+coronal] / \_\_ [+voice]\ (optional)$$

$$R_2 : [+liquid] \rightarrow \emptyset / [+coronal] \_\_ [+coronal, +strident]\ (obligatory)$$

Figure 2.2: Random rule set example

2.4.2 Mutation

An individual may undergo mutation by using the existing mutation operators used by the simulated annealing learner in Rasin et al. (2015). The full list of mutations is given in Appendix A. Each individual undergoes mutation with probability P_m, called the *mutation rate*. One of the hypothesis' components - lexicon or rules set - is selected with uniform probability (0.5), and a random mutation is applied to the selected component.

2.4.3 Crossover

Two hypotheses are mated by applying crossover either to their lexicons or their rule sets. The crossed-over component is selected at random with uniform probability.

2.4.4 Lexicon Crossover

The crossover operator for HMMs needed to be implemented in a meaningful way so that it will result in inheritance of valuable traits. The fact that HMMs are made of two layers of information - transitions and emissions - makes it important to develop a crossover operator that will capture both. Furthermore, the

implementation needs to protect the innovations learned in structure from being destroyed during crossover.

The implementations described below are inspired by two categories of previous works: those designed specifically for HMMs and others designed for mechanisms resembling HMMs, mainly graphs.

Relatively little work has been done trying to evolve HMMs directly using genetic algorithms. Yada et al. (1994), Won et al. (2004), Won et al. (2006), and Won et al. (2007) have used genetic algorithms for learning HMM topologies for DNA sequence analysis, using a graph representation similar to the one in section 2.4.4.2 below. Chau et al. (1997) used a transition matrix representation similar to the one described in section 2.4.4.1 below.

Since HMMs can be seen as a specific case of a directed graph, we also looked at implementations for evolving graphs. In this domain a large body of work has been done to evolve efficient neural networks topologies, which are represented as directed weighted graphs. Stanley and Miikkulainen (2002) offers a good overview of graph representations for evolving neural networks, mentioning crossover should implement some form of subgraph switching to preserve meaningful components of the graph. Variations of subgraph crossover are implemented in 2.4.4.2 and 2.4.4.3.

2.4.4.1 Transition matrix crossover

Each parent HMM is first transformed to a transition matrix. Row i in the matrix represents the transitions of state i in the HMM, so that entry $A_{i,j}$ is 1 if there

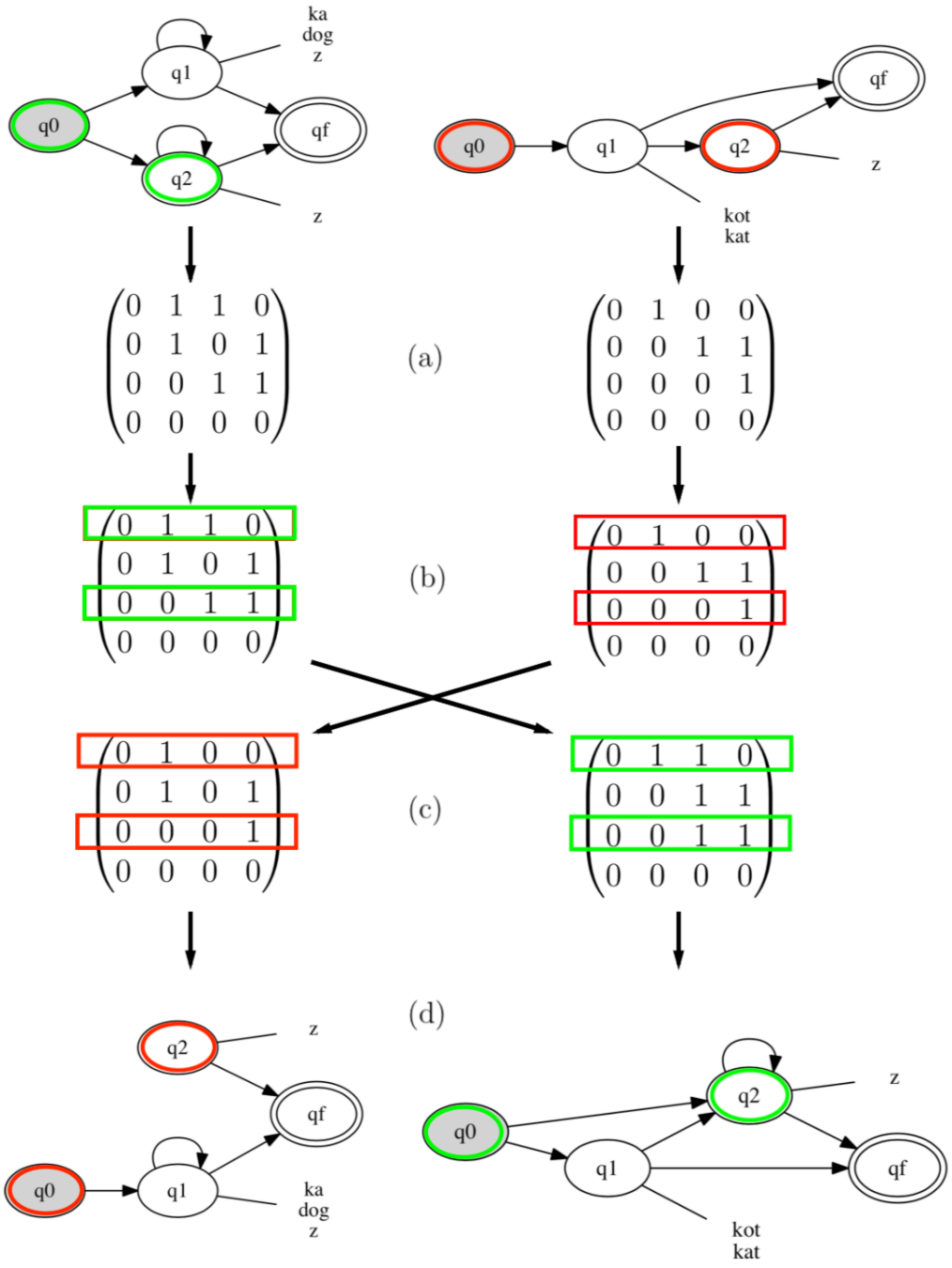


Figure 2.3: Transition matrix HMM crossover. (a) The transitions of each parent HMMs are represented as transition matrices. (b) Rows 0 and 2 are selected at random for crossover. (c) New transition matrices are created by switching the selected rows. (d) The offspring HMMs are constructed from the new transition matrices.

is a transition in parent HMM A from state i to state j. During crossover, the values from row i in each matrix may be copied to the other matrix's row i and vice versa, with uniform probability (0.5). If one matrix is larger than the other, rows can be removed from the larger one and added to the other. Each row moves with its respective state's emissions. The offspring HMMs are then re-constructed from the result transition matrices and emission lists.

This implementation is in the spirit of the canonical binary-representation crossover, in that it flattens the HMM structure to binary matrices that are similar to flat bit-strings, in hope that the offspring will be better than their parents. This may often prove destructive to the HMM, since it ignores its overall structure which may bear more meaning than a single state does. For example, at an intermediary stage of the learning process, a HMM which is not yet optimal often contains a loop of transitions, which concatenates segments into correct morphemes. This is not an optimal state because it is wasteful in terms of D:G, but it still leads to hypotheses that can fully represent the corpus. Since the rows are switched randomly during this crossover implementation, the loop may be broken, the HMM will fail to generate some morphemes and the hypothesis will become invalid. The following implementations try to cross-over the HMM while better keeping its structure.

2.4.4.2 Connected components crossover

Each parent HMM is represented as a directed graph, and its connected components are found using a variant of the canonical algorithm for finding connected

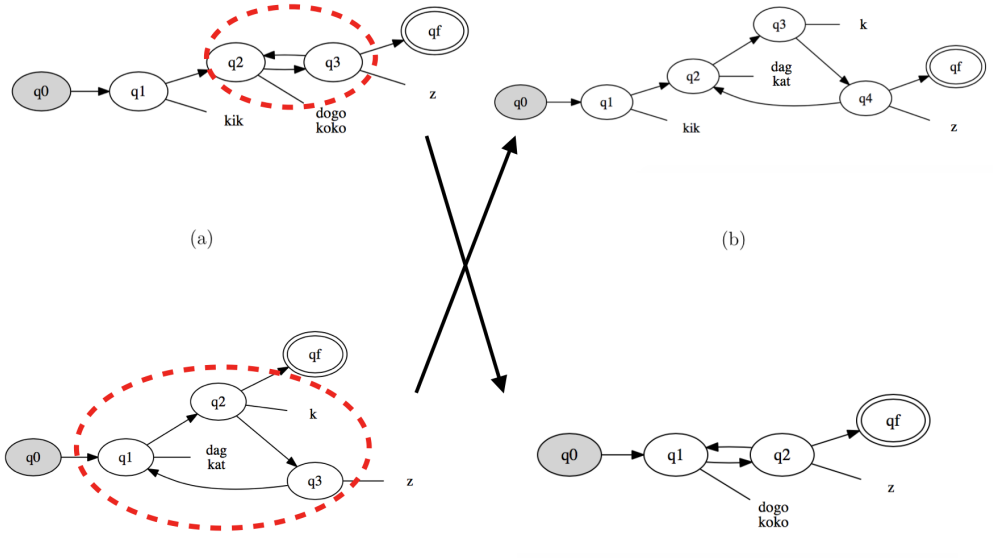


Figure 2.4: Connected component HMM crossover. (a) A connected component is selected from each parent. (b) Each offspring contains the switched over component from the other parent. Note how loops are preserved.

components, described in Pearce (2005). Two components are randomly chosen from each HMM, and switch places. One incoming arc and one outgoing arc from each component are kept, and are re-attached to the new inserted component. The emissions of each crossed-over state move with their respective states and stay intact. The motivation for this crossover operator is that it switches between groups of consecutive states instead of randomly selected ones; since the learner often needs to learn such consecutive transitions, e.g. stems and suffixes, this operator may help preserve these structures. At the same time it prevents loops from being broken, which may also preserve valid hypotheses.

2.4.4.3 Subgraph crossover

Subgraph crossover is based on the crossover operator used in Genetic Programming (GP), a family of evolutionary algorithms that operate similarly to genetic algorithms but are aimed at evolving programs. Programs are often represented as tree graphs, in which each node contains a functional operator (e.g. ”plus”, ”minus”). A tree thus makes up a program whose order of operations is set by the tree flow. The common crossover operator on GP trees consists of selecting a branch in each parent tree, cutting the sub-trees stemming from the selected branches, and hanging each sub-tree back at the other parent's cut branch. This implementation respects the learned functionality of the tree and has the potential of importing functionality from both parents to create superior offspring.

In our implementation for HMMs, an arc in each parent HMM is randomly chosen. The HMM is cut at the chosen arc, and the subgraph stemming from it is

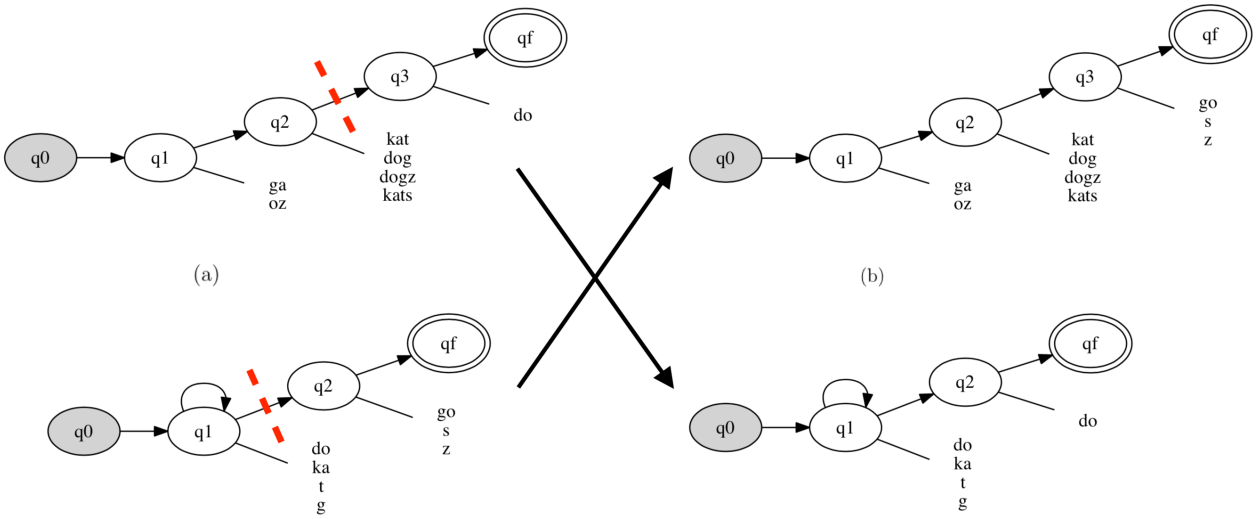


Figure 2.5: Subgraph crossover. (a) A transition arc in each parent is selected as a cutoff point - for the upper parent the arc from q_2 to q_3 and for the lower parent from q_1 to q_2. (b) The offspring are constructed by crossing over the parent graphs from above and below the cutoff points

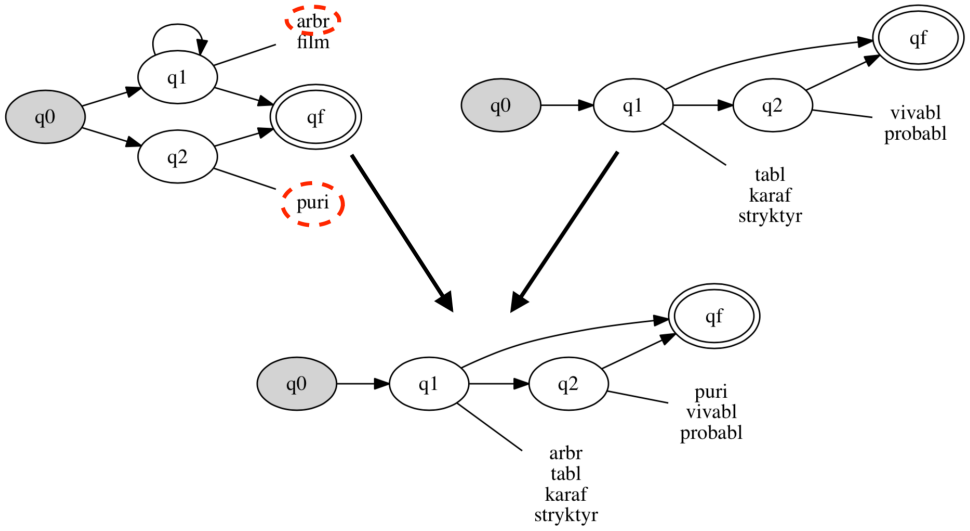


Figure 2.6: Unilateral emissions crossover. Emissions are randomly chosen from one parent and added to the equivalent states in the other parent. No emissions are deleted from either parent.

switched with the subgraph from the other parent. Since a HMM is not necessarily a tree (it may contain cycles), only arcs that serve as incoming/outgoing arcs of a connected component are selected as cutoff places. The emissions of each crossed-over state move with their respective states.

2.4.4.4 Unilateral emissions crossover

The HMM crossover implementations described above might turn out destructive for the learned grammar. First, because the segmentation and emissions are highly co-dependent, switching only parts of them may render the HMM useless. Second, the learned segmentation may be idiosyncratic to the specific grammar, where it interacts with the rule set. Crossing-over both the emissions and the

segmentation might prove destructive more often than not.

It may therefore be valuable to only cross-over the emissions learned in one parent to the other, hoping that the accepting parent, who had hopefully already learned a somewhat-correct segmentation, would benefit from morphemes it still hasn't discovered. No emissions are removed from either parent and the states transitions stay intact.

This crossover implementation leaves the evolution of HMM topology to the mutation operator alone. A non-destructive crossover that evolves both segmentation and emissions of the HMM may be though of for future development.

2.4.5 Rule Crossover

Like for lexicons, crossover for rules needed to be implemented in a way that will preserve useful rules learned throughout the evolution process. At first, a naive crossover was implemented that was allowed to cross-over rule internal parts, i.e. feature bundles. This proved more destructive since rules bear meaning as one unit containing change and context, and lose this meaning when these parts are separated or switched. The implementation was then changed to two operators that manipulate whole rules instead of parts of them.

2.4.5.1 Rule pair crossover

The two rule sets from both parents are aligned so that each rule is paired with the rule at the same position in the other parent. The rules in each pair then switch places with uniform probability (0.5). If one set has more rules than the other, the

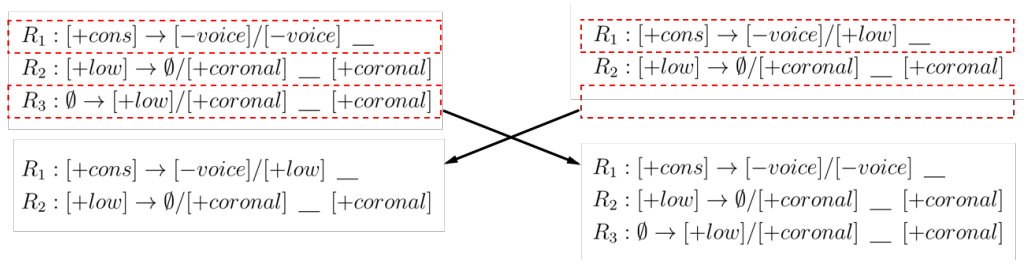


Figure 2.7: Rule pair crossover. Rules R_1 and R_3 are selected from each parent for crossover. Since the right parent has only two rules, it only receives R_3 from the other parent.

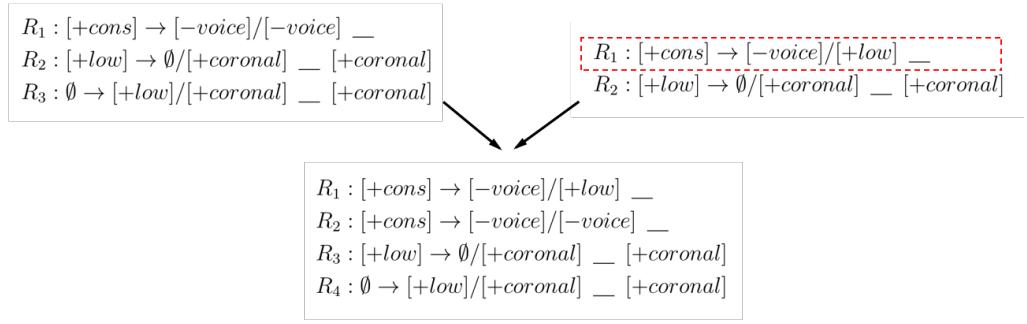


Figure 2.8: Unilateral rule crossover. Rule R_1 is randomly chosen from the right parent and added to the left parent at the same position, without removing any rules.

smaller one can receive rules from the larger. The overall number of rules stays the same.

2.4.5.2 Unilateral rule crossover

Following the same logic for unilateral HMM crossover, this operator adds rules from one set to the other, without removing any rules from either set. This assumes that removing rules from grammars might render them invalid; instead, it may be better to add rules that could prove useful, and let redundant rules be removed by mutation later.

2.4.6 Selection

A selection method's role is to receive the current population as input, and to output individuals that will make the population of the next generation. The function may output some individuals more than once, i.e. selection with repetition, which essentially means 'killing' under-performing individuals and favoring fitter ones.

Many selection methods have been proposed in the genetic algorithms literature. We implement two of the most common ones, *rank-based selection* and *tournament selection*, which offer a good balance between population diversity and selection pressure.

2.4.6.1 Rank-based selection

Rank-based selection is a variant of *fitness-proportionate roulette-wheel selection* (Goldberg, 1989), a method that assigns a selection probability to an individual based on its fitness value. For a population of size N, an imaginary roulette wheel is divided into N slices, each relative in size to each individual's fitness divided by the average fitness. The wheel is then "spun" N times, and each spin selects an individual for the next generation (with repetition). Since individuals are selected based on their fitness, better-performing individuals will get chosen more often, while worse ones will have less chance of survival.

Fitness-proportionate selection might create an unwanted selection bias in situations where a few individuals have superior fitness values compared to the rest of the population. These individuals will almost always be selected for survival,

and will quickly come to dominate the population, even though they are outnumbered. This can result in a decrease in population diversity and in premature convergence, which may prevent reaching the optimal solution.

Rank-based selection (Baker, 1985) mitigates this problem by assigning a selection probability based on an individual's rank in the population, instead of its direct fitness value. This leaves the selection function monotonically increasing, but relaxes the selection pressure by allowing less-performing individuals to survive, thus maintaining population diversity.

The best individual's rank is N, and the worst rank is 1. The individuals are first sorted by fitness values, and each individual is given a slice of the roulette wheel with size proportionate to its rank. The selection process then follows the same method as in roulette-wheel selection.

A linear scaling of each individual's rank is used, and allows to adjust the selection pressure (Baker, 1985; Razali et al., 2011):

$$Rank(n) = 2 - SP + \left(2 \cdot (SP - 1) \cdot \frac{(n-1)}{(N-1)}\right) \tag{2.1}$$

where N is the population size, n is the individual's absolute rank in the population (from 1 to N), and $Rank(n)$ is the scaled rank of an individual. SP, for Selection Pressure, is a configurable constant in range $[1.1, 2.0]$ that can be used to adjust the variability in scaled ranks. Higher SP will result in selection closer to fitness-proportionate selection, while $SP = 1.1$ will result in almost random selection.

2.4.6.2 Tournament selection

Rank-based selection may impose a heavy computational cost, since it requires the entire population to be sorted prior to selection. When population size increases, the sorting phase, executed for each generation with a cost of at least $O(NlogN)$, can slow down the search significantly.

Tournament selection is a common selection method that lowers the computational complexity by stochastically approximating more exhaustive selection methods. Like other methods, the tournament method selects N individuals from the population with replacement. A configurable parameter t is called the *tournament size*, and N tournaments are held to select N individuals. In each tournament, t individuals are randomly chosen from the original population, and the fittest of them is passed to the next generation. Tournament selection thus lowers the selection cost to $O(N)$.

Larger t means higher selective pressure. The case $t = 1$ is equivalent to randomly selecting N individuals with replacement. A common choice for tournament size that balances between selective pressure and population diversity is $t = 2$. Tournament selection is however still more prone to genetic drift and premature convergence than rank-based selection, since the lowest-performing individuals have a low to zero chance of survival. Tournament selection's origins and variants are discussed in detail in Goldberg and Deb (1991).

2.4.7 Fitness

The fitness of an individual is set to the hypothesis' encoding length (number of bits in $|G| + |D : G|$), as described in section 1.2.2. However, while the simulated annealing learner only considered hypotheses that describe the data in full, the genetic algorithm also assigns a fitness score to hypotheses that cannot represent the data.

This is because the simulated annealing learner starts the simulation with a naive over-generalizing hypothesis that describes the data, and slowly forms it into a more specific one; the genetic algorithm learner is launched with a large population of random hypotheses, which most likely don't represent the data at all. For genetic algorithms that operate on non-continuous hypotheses spaces, like our learner, it is common practice to design the fitness function so that even invalid hypotheses receive a fitness score, usually with a penalty, to help guide the search towards meaningful hypotheses (Mitchell, 1998b).

A hypothesis that fails to fully represent the corpus is thus assigned a fitness value according to the following heuristic.

Hypothesis Fitness Calculation

- A valid hypothesis is assigned a fitness value equal to $|G| + |D : G|$
- An invalid hypothesis that doesn't represent all words in the corpus gets a penalized fitness value equal to:

$$\textit{threshold} + (\textit{penalty} \cdot \textit{number of not-represented words})$$

- $penalty$ is a configurable cost in bits to add for each not-represented word, e.g. 1,000 bits.

The $threshold$ is set according to the worst hypothesis in the population:

- If the entire population fails to parse all words, the threshold is set to a high value (e.g. 1,000,000), higher than any reasonable hypothesis energy.
- If a hypothesis exists that can fully represent all words in the corpus, the threshold is set to the energy of the worst valid hypothesis.

This heuristic will always score invalid hypotheses as worse than valid ones, but orders them by their potential validity. This helps to quickly guide the search towards hypotheses that represent the data, when the population has just been randomized and most of the hypotheses don't represent any data. Its disadvantage is that it requires the parsing of all words in the corpus, while the SA learner in Rasin et al. (2015) stopped the process upon the first word that couldn't be parsed. Having to always parse all words slows down the search. An optimized version of this

heuristic, for example applying it only during an initial bootstrapping phase of the algorithm, can be thought of for future work.

2.4.8 Elitism

Elitism is a simple yet effective mechanism that prevents losing the best individuals in a population due to crossover and mutation (De Jong, 1975). A configurable percentage of the best individuals in each generation are kept for the next generation, even if they went through mutation or crossover. The effect of elitism is tested in 2.4.12.

2.4.9 Parallelization

The major disadvantage of the previous learner's optimization method, simulated annealing, is that in its canonical form, it is a sequential algorithm that operates on a single hypothesis at a time, mutating it step by step. This caused simulations with relatively small corpora, of a few dozen words, to take days or even weeks to complete.

Attempts at parallelizing simulated annealing it have been made (Greening, 1990; Azencott, 1992; Onbaşoğlu and Özdamar, 2001; Zomaya and Kazman, 2010), and we have implemented two of the most common techniques: the periodically-interacting scheme, where several simulated annealing chains are run in parallel and exchange information periodically about the best hypothesis found thus far; and the multiple-trials scheme, in which a single simulated annealing chain is run,

with each decision step consisting of choosing between p neighbour hypotheses instead of 1, making use of p parallel processes.

These methods did not prove advantageous for our purposes, since they did not significantly improve the run time of the algorithm, and no qualitative difference was observed. As we will also witness with the parallelization of genetic algorithms below, this may have been due mostly to the fact that these parallelization techniques require sending single hypotheses back-and-forth between processes for evaluation. The overhead of this operation offsets the advantage which may have been gained by parallelization.

Unlike simulated annealing, genetic algorithms are intrinsically parallelizable. Since they can operate on large populations of hypotheses instead of a single hypothesis, in a multiprocess environment they can offer a potential speedup linear in the number of processes. This made genetic algorithm a good candidate for replacing the search method.

2.4.9.1 Naive parallelization

Since the three basic operations of the GA - mutation, crossover and fitness evaluation - operate independently on each individual (or pairs thereof), parallelization can be easily achieved by distributing these operations to multiple parallel processes: a simple 'master-slave' scheme can be used, in which a 'master' process distributes work to the other p 'slave' processes; the slaves perform mutation, crossover or evaluation on individuals, and the result is sent back to the master.

This technique should potentially result in a speedup linear in p. However,

it incurs an overhead cost of communicating each hypothesis back and forth between the main process and the workers. In our learner, experiments showed that this was not worth the parallelization gain, since the computation cost of the genetic operators is offset by the communication cost. Attempts at lowering the communication costs were made, mainly by sending only the parts of hypothesis that are relevant for each operator. However, since each of the operators - mutation, crossover, and evaluation - require most parts of the hypothesis to apply, this did not prove fruitful.

2.4.9.2 Island model

A less trivial parallelization scheme for genetic algorithms is the *Island Model* (Gordon and Whitley, 1993; Adamidis, 1994; Cantú-Paz, 1998), sometimes called a *Multiple-Deme* model. Inspired by evolutionary speciation and niching - the theory of how species evolve in separated environments - the model divides the overall population into *islands* or *demes*. Each island is in essence an independent basic genetic algorithm as described above, which evolves a subpopulation of the overall population. The islands are run on parallel processes that are mostly independent of each other, until once in every number of generations a *migration* step occurs, in which some individuals from each island are copied to an adjacent island. The migration is orchestrated by a parent process. After each island has completed the overall number of generations, the best hypothesis from all islands is taken as the optimal solution.

The island model offers significant advantages over the basic single-population

Algorithm 2 Island Model Genetic Algorithm

$I \leftarrow$ number of islands
$N \leftarrow$ single island population size
$M_i \leftarrow$ migration interval
$M_r \leftarrow$ migration ratio
$T \leftarrow$ migration topology function
$F \leftarrow$ fitness function

for each $i \in [1, I]$ **do in parallel**:
    initialize random hypotheses population P_i of size N
    $generation_i \leftarrow 0$

    while $generation_i <$ *max-generations* **do**:
        run Algorithm 1 (Basic Genetic Algorithm) for M_i generations
        send the best $M_r \cdot N$ hypotheses from island i to island $T(i)$
        delete the worst $M_r \cdot N$ hypotheses in island i and replace them with incoming migrants, if any
        $generation_i \leftarrow generation_i + M_i$
    end while
end for

return $\text{argmax}_{h \in \cup P_i} F(h)$

genetic algorithm, and over its naive parallelized version. First, the division of the population into islands allows to overcome the process communication overhead: the processes rarely communicate, and when they do, only a small fraction of the population is communicated between processes. This communication cost is subsumed by the overall simulation time, which is spent in parallel almost without interruption.

Apart from improving the run time performance, it was argued and shown (Whitley et al., 1999) that partitioning the overall population into isolated islands may also offer a qualitative improvement, by enabling *niching* to occur: each island may potentially explore different areas of the search space, while a single-population algorithm would have converged prematurely on a underdeveloped solution. The migration steps allow the islands to exchange their specialized innovations, which will hopefully merge to form the optimal solution. In our context, the isolated islands may potentially learn different parts of the correct hypothesis, e.g. some islands may discover parts of the correct lexicon, while others will discover the correct rules.

The rate with which the migration step occurs is called the *migration interval* ($M_{interval}$), and the percentage of individuals that migrate is called the *migration ratio* (M_{ratio}). Some decisions need to be made when implementing an island scheme, mainly how individuals are selected for migration, and how to choose the target island for migrants (the *migration topology*). The migration interval and ratio need to be selected with care: a too-frequent migration interval can cause the overall population to lose diversity too quickly, which would miss the benefit

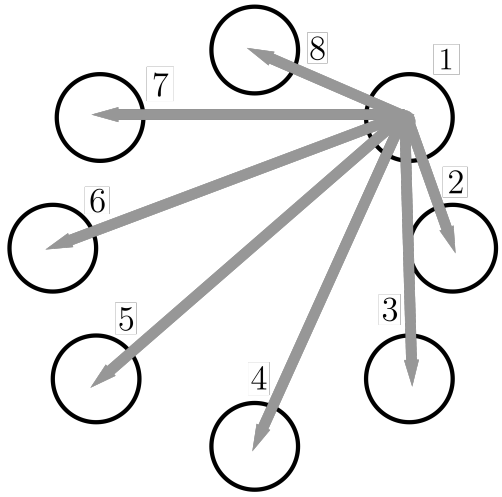


Figure 2.9: Round-robin migration topology for a single island. The destination island number is incremented by 1 with each migration step.

of the island model; a too large migration ratio might cause the same effect. On the other hand, high migration intervals and low migration ratios will prevent the islands from exchanging useful genetic information. The island model algorithm is given in Algorithm 2.

We implemented two migration topologies. The first is a naive circular scheme that always migrates individuals from one island its immediate neighbour, i.e., from island I to island $I+1$, or more exactly to $(I+1)mod(total\_islands)$. This strategy is easy to implement, but is very slow in propagating genetic novelties between populations: a trait discovered in one island will take many generations to propagate to other islands. It is also prone to bottlenecks that may be caused by island processes that are slower or have crashed.

We opted for a toplogy (Whitley et al., 1999) that also arranges the islands in

a ring, but performs migration in a round-robin fashion (figure 2.9): on the first migration step, each island sends its migrants to its immediate neighbour. On the second migration, the island destination is incremented by 1, and so forth. This topology allows all islands to exchange migrants with each other, enables genetic innovation to spread faster between islands, and helps to avoid bottlenecks.

Another implementation decision is how to select outgoing migrants from an island, and how to incorporate incoming migrants. We adopt the method in Whitley et al. (1999): the most fit M_{ratio} individuals of an island are sent as migrants, and the worst M_{ratio} are replaced by the incoming migrants.

Another technical decision to be made is whether to perform synchronous or asynchronous migration, i.e., whether all islands should wait for each other to complete a specific number of generations before a migration can occur. We followed the recommendations in Bennett III et al. (1999), mainly: the islands never wait for each other (asynchronous migration), since some islands may run slower than others; if incoming migrants are not available at the migration step, the island continues without incorporating any; and if an island is sent several groups of immigrants from different islands before its migration step, the most recent group of migrants is taken. These heuristics help to speed up the search, since almost no synchronization delays occur and the dependency between islands is minimized.

2.4.10 Technical information

The genetic algorithm was implemented on top of DEAP (Fortin et al., 2012), a Python framework that allows quick prototyping and implementation of evolutionary algorithms. Complex simulations with large corpora that require hundreds of multiple processes are run using Amazon Web Services (AWS) and Microsoft Azure cloud platforms, which enable distributing a single simulation over multiple machines. Migrations between islands are made through cloud storage (AWS S3).

2.4.11 Performance

The following table gives an overview of simulation run times of the previous simulated annealing learner compared with the genetic algorithm learner, on identical corpora and the same hardware. It can be seen that the new optimization method yields a significant speedup, lowering simulation times from several days to hours[2]. It should also be noted that the improved optimization not only speeded up the existing simulations, but also allowed to run new simulations with much more complex corpora - an order of magnitude larger, in fact - that could not have been tested earlier. Chapter 3 will present such a corpus.

[2]Note that the completion criteria for the two algorithms are different. Simulated annealing stops when the temperature reaches the bottom threshold (starting at 75 or 50 in this comparison), and the genetic algorithm stops after a configured number of generations (10,000 in this comparison). The displayed times are for time to completion. Time to convergence is significantly lower for the genetic algorithm, which also means that the configured number of generations can be lowered significantly.

Corpus	Words	Simulated Annealing	Genetic Algorithm
Morphology only	32	11 hours	2.5 hours
Voicing assimilation	32	33 hours	11 hours
Two rules interaction	105	168 hours	17 hours
Opaque rule ordering	105	167 hours	6 hours

Figure 2.10: Simulation time to completion

2.4.12 Hyperparameters comparison

Genetic algorithms are notorious for having many hyperparameters to configure, usually leading to tweaking the model by trial and error. Plus, optimal configurations are often task-specific, and there is no one rule of thumb for all genetic models. Performing a 'grid search' to find the optimal parameters is infeasible due to the large number of configurable parameters of the model. For this reason we conduct a limited comparison of the effect of prominent parameters of the model, to test their efficiency and how they affect the learning process.

All of the comparison were run using the smaller voicing assimilation corpus from Rasin et al. (2015) that contains 32 words, and use the following hyperparameters unless otherwise noted: 32 islands with population size 200 (total 6,400); crossover rate 0.2; mutation rate 0.8; tournament selection with tournament size 2; migration ratio 10%; migration interval 50; elite size 5%. All the figures below show the best fitness from all islands for each generation. Lower fitness is better.

2.4.12.1 Crossover and mutation rates

The crossover operators that were introduced to the learner in this work are still not expressive enough to evolve hypotheses alone. The model still relies heavily

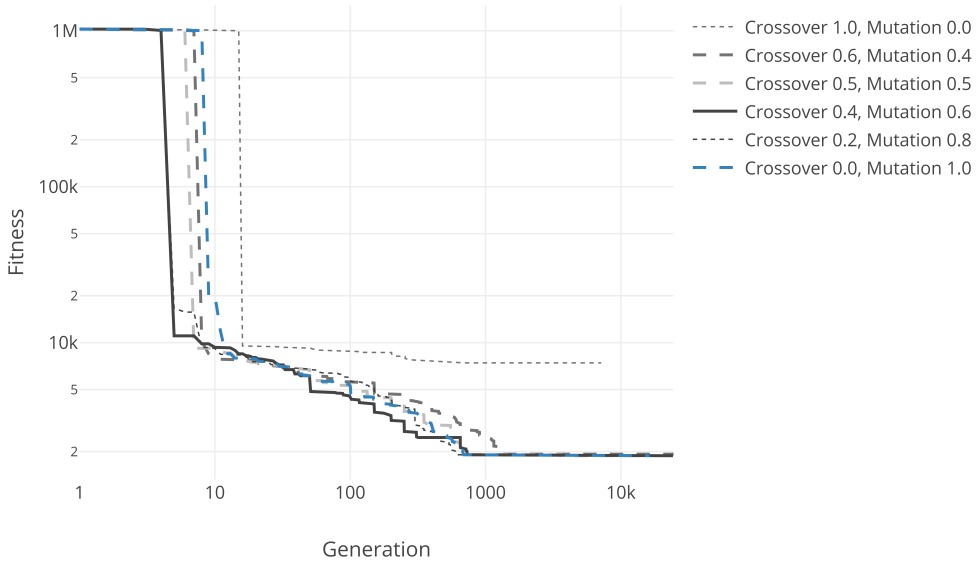


Figure 2.11: Crossover and mutation rates

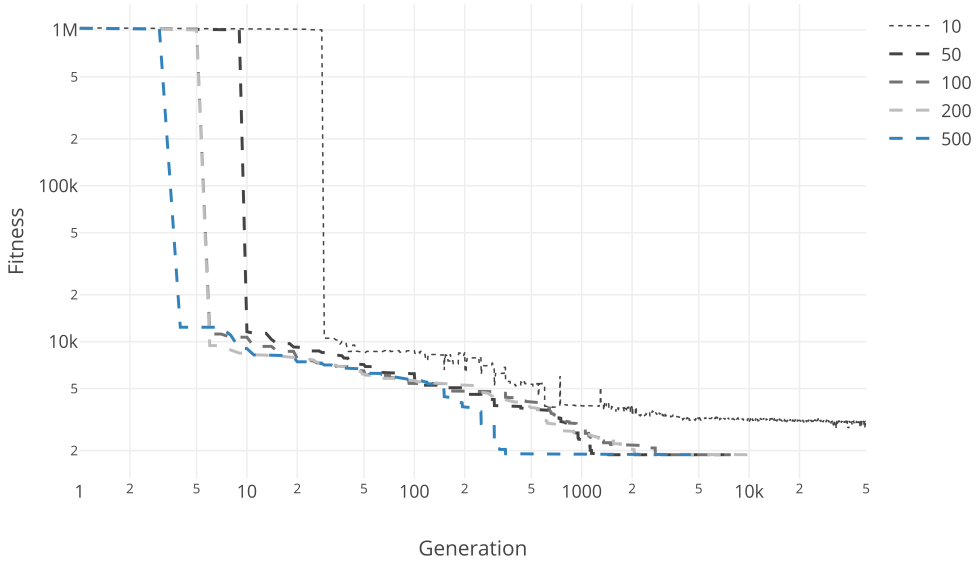


Figure 2.12: Island population size

on mutations, mainly in order to evolve the emissions in HMMs and to evolve the specifics of each rule. This can be seen clearly in figure 2.11: when the mutation rate decreases, convergence is delayed. When the model relies only on crossover, the simulation converges prematurely to the wrong hypothesis. However, it can be seen that some crossover is somewhat better than no crossover: the convergence without any crossover is delayed, though not by much - all choices lead to roughly similar convergence rates as long as mutations are allowed.

2.4.12.2 Island population size

The population size should be selected to be large enough to allow population diversity, but not too large so it won't slow down the simulation. Figure 2.12 shows how a population size too small (10) fails to preserve the diversity required to reach the correct hypothesis, and converges on a wrong one (the plot for the smallest population size oscillates because no elite is kept). It can also be seen that the largest population size also reaches the correct hypothesis, although being unnecessarily slow (the visualization doesn't reflect the slowdown since it plots against generations and not time).

2.4.12.3 Elite size

As mentioned in 2.4.8, elitism is used to preserve the best individuals of a population from being lost during mutation or crossover steps. It can be seen in figure 2.13 that without elite protection the fitness plot oscillates, i.e. the best individual is periodically lost. This did not prove fatal in this test for the rather simple voicing assimilation corpus, but crippled the search significantly for more complex corpora that run for many more generations. It should be noted that a large elite size (50%) has guided the search faster towards convergences in this case, but may cause premature convergence with more complex corpora that require a more refined search.

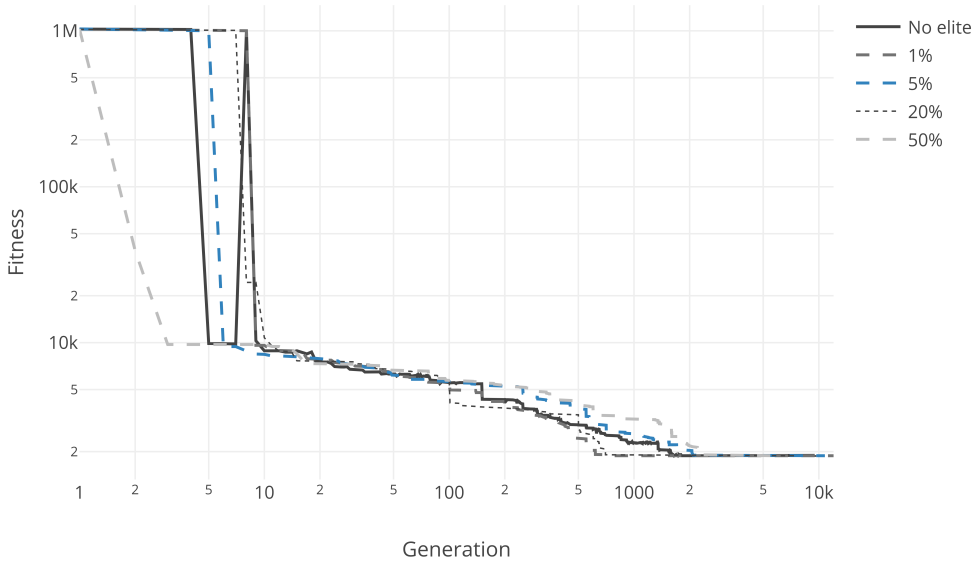


Figure 2.13: Elite size

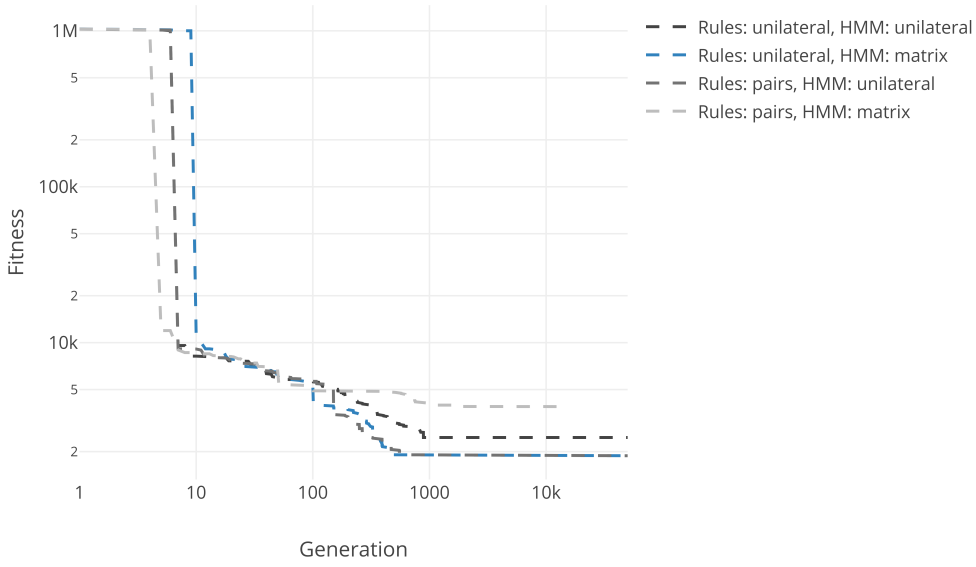


Figure 2.14: Rules and lexicon crossover operators

2.4.12.4 Crossover operators

For this comparison, both crossover and mutation rates were set to 0.5. Two crossover implementations of each component were tested: unilateral emissions crossover and transition-matrix crossover for HMMs; and unilateral crossover and rule-pairs crossover for rules (figure 2.15). It can be seen that combinations of unilateral crossover for one component and a bilateral crossover for the other worked best, and converged on the correct hypothesis. Other combinations of all-unilateral crossovers or all-bilateral crossovers failed to converge. The combinations that worked best were adopted for future simulations, although we couldn't find an explanation for this behavior. We leave its investigation for future work.

2.4.12.5 Selection methods

Rank-based selection and tournament selection were tested against each other with different parameters. As mentioned in section 2.4.6, tournament selection with tournament size 1 and rank selection with SP close to 1.1 constitute a totally random selection. This can be seen in figure 2.15 where simulations with these parameters did not converge correctly, demonstrating the importance of selection pressure.

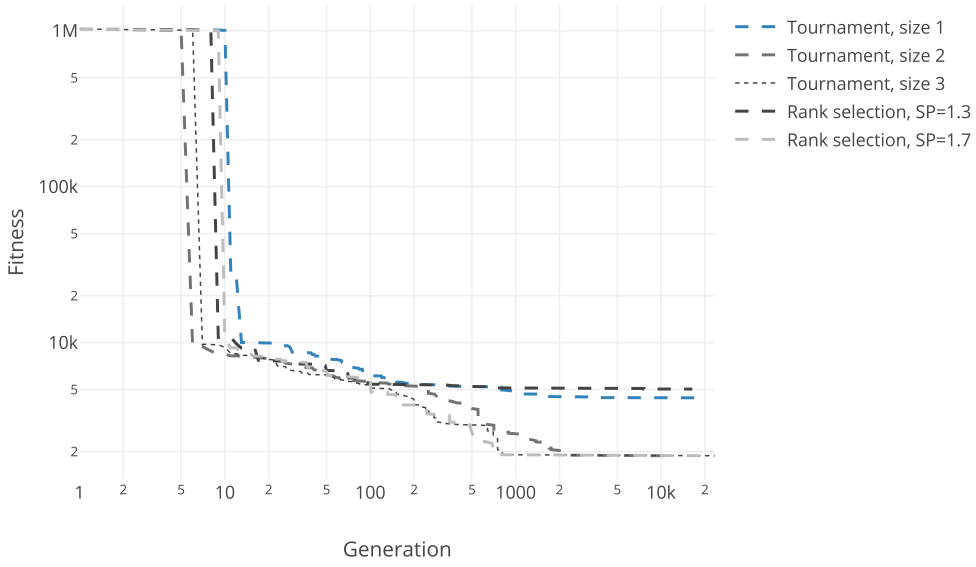


Figure 2.15: Selection methods

Chapter 3

French rule interaction

3.1 Background

The following chapter is largely based on a phonological process in French described in Dell (1981). French has a rule that deletes word-final liquids if they are preceded by a non-sonorant obstruent, given in (3.1). The rule is optional, so for example, the underlying form /tabl/ ('table') can surface as both [tab] and [tabl], and /arbr/ ('tree') can surface both as [arbr] and [arb]. Since liquids are not deleted after sonorants, /yrl/ ('yell') can surface as [yrl] but not as *[yr], and /parl/ ('speak') can surface as [parl] but not as *[par].

$$\begin{bmatrix} +liquid \end{bmatrix} \rightarrow \emptyset / \begin{bmatrix} +cons \\ -son \end{bmatrix} \_\_ \text{ (optional)} \tag{3.1}$$

This rule on its own poses a non-trivial challenge to the child learner. Since

the child is presented with only positive examples by an adult who internalized (3.1), they may equally reach an over-generalizing rule (3.2), that posits deletion of liquids after any consonant, including sonorants.

$$\left[+liquid\right] \rightarrow \emptyset / \_\_ \text{ (optional)} \tag{3.2}$$

While (3.2) fits the data equally well, French speakers reach the more specific (3.1), which restricts the deletion environment. If the child learner followed the SPE evaluation metric, they would not have reached the correct, more restrictive rule, because its description is longer: specifying a more restricted environment requires more symbols. Instead, they would have arrived at (3.2), the over-generalizing rule, which produces malformed utterances such as *[par]. In addition, the child learner is not given negative cues ruling out illegal forms such as *[yr], which would have directed them towards the more restricted environment.

At the other extreme, a learner using the subset principle as an evaluation metric would face another problem: undergeneralization. If the child is not exposed to both forms of each word produced by (3.1), the subset principle would direct the learner towards a grammar that memorizes observed forms instead of generalizing, e.g. storing both [tabl] and [tab] in G instead of collapsing them, since it produces a language that is a subset of the language produced by 3.1. For example, if the learner happens to hear [ivr] ('drunk') but never its L-deleted form [iv], then the language produced by (3.1) would be a superset of the language

producible from a memorizing lexicon, and (3.1) will not be chosen.

Rasin et al. (2015) have shown that the MDL-based learner presented in Chapter 1 reaches the correct optional liquid deletion rule, by balancing between the grammar economy metric (SPE) and the subset principle metric. An MDL learner is able to induce the correct optional rule since it does not fall into either pitfalls of these metrics. Under MDL, the SPE metric's penchant for over-generalization is met with a cost: (3.2) will have a shorter $|G|$, but will have to pay it back in $|D{:}G|$ for every exception like *[par]. Similarly, the subset principle metric will yield a shorter $|D{:}G|$ since it memorizes exceptions in G, but this will cause $|G|$ to grow significantly.

When tested on a French corpus governed by the rule in (3.1), the MDL learner in Rasin et al. (2015) reached a grammar that contained the correct lexicon and deletion rule, by collapsing pairs like [tabl] and [tab] in order to minimize $|G|$ and restricting the deletion environment in order to minimize $|D{:}G|$. However, due to performance limitations, the learner was not tested on the full phonological phenomenon described in Dell (1981), given below.

3.1.1 Rule interaction

In addition to the optional L-deletion rule, French has an optional rule that inserts schwa at the end of a word that ends with two or more consonants, if the next word begins with a consonant:

$$\emptyset \rightarrow \text{ə} / CC\_\#C \text{ (optional)} \tag{3.3}$$

For example, *parle mal*, 'speaks badly', can be pronounced either [parlmal] or [parləmal]; and *film pourri*, 'lousy film', can be pronounced either [filmpuri] or [filməpuri].

The schwa epenthesis rule (3.3) interacts with the L-deletion rule (3.1), and their ordering is important: the epenthesis rule precedes the deletion. This creates a 'bleeding' relationship - inserting the schwa destroys the environment where L could have been deleted, since it is no longer at the end of the word. For example, the following surface forms are generated when the rules are ordered correctly:

1. rompre mal [rompr(ə)mal] 'break-up badly'

2. couvercle sale [kuverkl(ə)sal] 'dirty lid'

3. arbre puri [arbr(ə)puri] 'rotten tree'

For the same underlying forms, if deletion was applied before epenthesis, both rules could operate in series and the following ungrammatical surface forms would be generated:

1. rompre mal *[rompəmal]

2. couvercle sale *[kuverkəsal]

3. arbre puri *[arbəpuri]

Note that the rule ordering becomes significant when the first word ends with three consonants, of which the last one is a liquid. In that case, if the liquid is deleted by (3.1), the environment would still allow for (3.3).

The interaction between the optional L-deletion and schwa-epenthesis rules poses a non-trivial challenge for the learner. First, as mentioned above, the optional L-deletion rule alone is sufficient to rule out both the SPE grammar economy metric and the subset principle.

Secondly, with the combination of two optional rules, the learner faces a compound challenge: an opaque rule ordering, where both orderings are consistent with the observed data. Since the two rules are optional, both orderings (epenthesis before deletion and vice versa) generate the observed surface forms; the difference between the orderings is in the redundant, ungrammatical surface forms that are generated by the incorrect ordering, given above. The learner never faces these forms since it is never met with negative evidence, yet the correct ordering needs to be learned. A learner following the economy metric will fail to prefer either ordering since it has no restriction on the generated surface forms. Once again this calls for an evaluation metric that balances grammar economy with data restrictiveness. Moreover, the economy metric has no mechanism for specifying rule ordering since it only considers the number of symbols in rules, ignoring their order. Since in this case both orderings have the same number of symbols, no ordering will be preferred.

The learner thus needs to not only generalize beyond the data and to conclude that pairs like [arbr]-[arb] need to be collapsed, but also learn rules through the opaque surface, learn that they are optional, and that their order is significant. To the best of our knowledge, no general solution to these challenges has been provided in the literature, let alone to this specific French process. In the following

section we show how a learner using MDL as an evaluation metric is able to reach the correct rule ordering and lexicon for the French rule interaction.

3.2 Simulation

The corpus for the simulation is based on examples from Dell (1981) and additional French words. The target lexicon contains 26 stems and 9 suffixes, including the null suffix:

Stems	tabl, arbr, mordr, parl, film, kuverkl, yrl, klop, kylt, provok, prut, klad, krab, burk, kurb, kapt, kupl, odor, amur, karaf, furyr, byl, batir, purpr, filtr, rompr
Suffixes	puri, mal, byvabl, fad, timid, kif, abil, ivr, $\emptyset$

The data presented to the learner was the output generated by all possible lexicon combinations with all possible rule applications. Having two optional rules in the underlying grammar produces up to 3 surface forms for each stem-suffix combination in the lexicon (and not 4 since the rules are in a bleeding relationship). This created 479 surface forms in total, a corpus significantly larger than previously tested (more than twice the largest corpus in Rasin et al., 2018). The improvement in performance gained by using the genetic algorithm enabled us to test the learner on this scale for the first time. The full corpus is given in appendix B.

In order to facilitate the search, an extra non-standard feature $\left[\pm center\right]$ was used to represent schwa, so the search could reach the required feature bundle

more easily and not have to mutate until it reached this representation: $\begin{bmatrix} -low \\ -high \\ -front \\ -back \end{bmatrix}$.

3.2.1 Results

The simulation was run using the genetic algorithm with 750 islands that had a population of 200 hypotheses each (total population 150,000). The simulation ran for 5,000 generations and was stopped after all islands converged on the same hypothesis and didn't improve for more than 1,000 generations. We used three AWS c5.18xlarge machines with 72 vCPUs each (3.0 GHz Intel Xeon), with each machine running 250 islands.

The final hypothesis is presented in Figure 3.1. First, it can be seen that the learner induced R_1 - an optional schwa epenthesis rule that operates at word boundary when the next word starts with a consonant. This almost matches the target epenthesis rule governing the corpus, apart from the left context environment which should have limited the application to words ending in two consonants. The missing environment is explained below.

R_1 is followed by two deletion rules, R_2 and R_3. It can be seen that with R_2, the learner has found a different way to express the underlying behavior "L is deleted only if schwa was not inserted": since the left context of R_2 is the correct context for the target R_1 (two consonants), and the right context for R_2 is $[-voice]$ and schwa is voiced, R_2 can only apply if R_1 didn't. The over-restricted

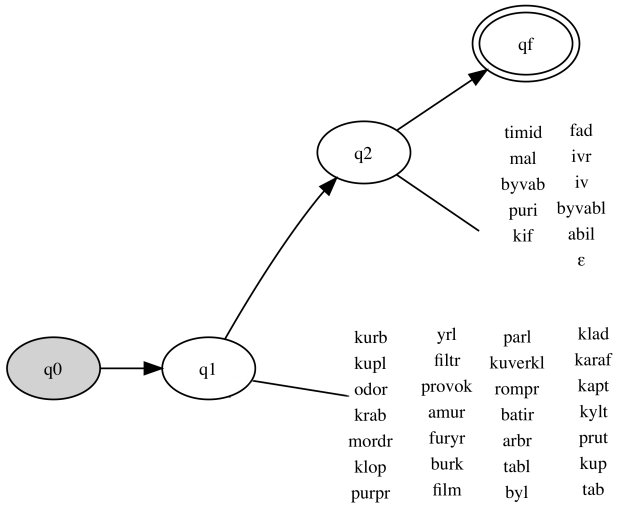
$G_{final} =$ {

RULES: $\boldsymbol{R_1 : \emptyset \rightarrow \text{ə} / \_\_ \#[+cons]}$

$\boldsymbol{R_2 : [+liquid] \rightarrow \emptyset / [+cons][+cons]\_\_[-voice]}$

$R_3 : [+strident] \rightarrow \emptyset / \_\_[+cons][-center][-voice]$

LEXICON:



}

Description length: $|G_{final}| + |D{:}G_{final}| = 1,310 + 4,514 = 5,824$

Figure 3.1: Final grammar for the French simulation. The grammar includes schwa epenthesis and liquid deletion rules (in bold) and a redundant deletion rule that has no effect on the phonological mapping.

$$G_{final'} = \begin{cases} \text{RULES:} & R_1 : [+liquid] \rightarrow \emptyset / [+cons][+cons]\_\_[-voice] \\ & R_2 : [+strident] \rightarrow \emptyset / \_\_[+cons][-center][-voice] \\ & R_3 : \emptyset \rightarrow \text{ə} / \_\_\#[+cons] \\ \text{LEXICON:} & \text{same as in } G_{final} \end{cases}$$

Description length: $|G_{final'}| + |D{:}G_{final'}| = 1,310 + \mathbf{4{,}695} = 6,005$

Figure 3.2: Final grammar with same lexicon and reversed rule order. The description length is worse because of $|D{:}G_{final'}|$

environment of R_2 does influence the lexicon though, and leads to suboptimal segmentation: it can be seen for example that [tabl] and [tab] were not collapsed in the final lexicon. This increase in lexicon size is offset however by the savings in data description length, explained below. Apart from such pairs, the lexicon was segmented correctly.

Although R_2 is in practice restricted to the correct target context, R_1 is more general than expected, and the final grammar can generate ungrammatical forms such as *[amurəpuri]. This should have resulted in an encoding length worse than the target length, due to the $|D{:}G|$ term, but the final grammar encoding length is 110 bits shorter than the target.

This brings us to the role of R_3: this rule does not have any effect on the phonological mapping - an identical grammar that does not contain it generates the same surface forms. Even so, removing it from the grammar results in an increase of 225 bits in encoding length, i.e. the redundant rule compounds with the suboptimal rules to lower the total encoding length, although it has not practical effect.

This is consistent with a behavior we saw in previous simulations. It seems that the transducer composition process, used to build the final grammar transducer, is introducing artifacts that influence the final encoding length, due in part to the naive encoding scheme we are currently using. We are currently investigating this and believe that using a less naive encoding scheme (e.g. entropy encoding) will solve this issue [1]. We are also testing a version of the learner which minimizes the final grammar transducer to its minimal equivalent form, which seems to eliminate such artifacts. We also believe that enlarging the corpus size may make the learner less prone to such artifacts, since as the underlying lexicon grows, fewer accidental gaps are likely to exist in the corpus.

It should be especially noted that although the learner did not finalize the rules to the correct environments, it did learn the correct ordering of epenthesis before deletion. To verify that this was not a search mishap, we checked the encoding length of the same hypothesis that has a reverse rule order (epenthesis after deletions). The description length for this hypothesis is given in 3.2 and is worse by almost 200 bits. This is because, as expected, the reverse grammar also generates ungrammatical forms such as *[arbepuri], that add to $|D{:}G|$. Similarly to the single L-deletion corpus, where the $|D{:}G|$ term encouraged the learner to restrict the environment, in this corpus the term also helped push the learner towards the correct rule ordering.

[1] See Rasin et al. (2018) where similar artifacts were observed and for a discussion of the influence of the encoding scheme on the final hypothesis

Chapter 4

Discussion

In this work we have improved upon the MDL-based learner from Rasin et al. (2015) in two respects. First, the simulated annealing optimization method was replaced by a more powerful genetic algorithm that enabled us for the first time to test the learner on larger and richer corpora. Up until now, the learner was only tested on toy datasets with a limited number of words and segments.

Secondly, the improvement in performance enabled us to examine a complex phonological phenomeon, based on French rule interaction, that compounds several learning challenges - optionality and opaque rule ordering. The improved learner showed promise in learning these in tandem. To our knowledge, this is the first proposed learner to achieve this task.

There is, however, more work to be done for future research. For example, the current learner is essentially a *batch learner*, in that it always parses the complete corpus as a whole. A modification to make it an *online learner*, that receives

updates to the corpus as it progresses, can be though of for future work to make the learner more psychologically plausible. Another modification that may make the learning process more realistic is to enable the learner to mark certain words as exceptions or invalid, thus supporting other theories of lexicon representation and at the same time simulating learning from noisy data.

There is also more work to be done before the learner can face true natural language corpora. On the technical front, the major setback of the learner is still its run time, which is highly influenced by the operators used to compose the grammar. Transducer composition, which is used extensively in the grammar generation and parsing process, is an expensive operation, that becomes slower as the corpus size increases. Transducer composition is currently the major bottleneck in the simulation run time, and it also introduces artifacts to the final grammar as seen in the previous chapter.

We believe however that these are technical difficulties related to the current choices of representations and encoding, and that they do not lessen the theoretical promise that the MDL principle, as a proxy for simplicity, offers as an evaluation metric.

Appendix A

Mutations list

This appendix describes all possible mutations used to generate a variant of a hypothesis.

A.1 Mutations on HMM

1. Combine emissions: pick two emissions at random, concatenate them, and add the result to a random state

2. Clone emission: pick an emission at random and add it to a random inner state

3. Move emission: pick an emission at random, remove it from its state and add it to a random inner state

4. Advance emission: pick a random state q_1. From q_1, pick an emission and

an outgoing state q_2 at random. Create a new state: q'. Add the chosen emission to q'. Remove the chosen emission from q_1. Add the transitions: q_1 to q', q' to q_2, and q' to q'.

5. Add state: add an empty state to the HMM (with no emissions or transitions)

6. Remove state: remove a random state, all its emissions and all arcs connected to it

7. Add transition: add a new transition between two random states (chosen with repetitions)

8. Remove transition: remove a random transition from a random state

9. Add segment to emission: add a random segment from the segment table to a random emission in a random position

10. Remove segment from emission: remove a random segment from a random emission

11. Change segment in emission: replace a random segment from a random emission with a different random segment

12. Add emission to state: add a random segment from the segment table as a new emission to a random state

13. Remove emission from state: remove a random emission from a random state

A.2 Mutations on feature bundle list

1. Add feature bundle: create a random feature bundle and insert it in a random position in the list
2. Remove feature bundle: remove a feature bundle at a random position
3. Change existing feature bundle: create a random feature bundle and mutate it using one of the mutations on feature bundles:
   (a) Add feature: add a random feature with a random value to the feature bundle
   (b) Remove feature: remove a feature at random from the feature bundle
   (c) Change feature value: flip the value of a random feature

A.3 Mutations on rule set

1. Add rule: generate a random rule with random feature bundles in each of the 4 parts of the rule: *change*, *focus*, *left context*, and *right context*. Add the rule to the rule set
2. Remove rule: remove a random rule from the rule set
3. Demote rule: pick a random rule. Move it down in the rule order
4. Change rule:
   (a) Mutate focus: mutate the *focus* feature bundle

(b) Mutate change: mutate the *change* feature bundle

(c) Mutate left context: mutate the *left context* feature bundle list

(d) Mutate right context: mutate the *right context* feature bundle list

(e) Mutate obligatoriness: flip the value of the obligatory value (which determines whether a rule is optional or obligatory)

Appendix B

French simulation data

Corpus data: amur, amurabil, amurbyvab, amurbyvabl, amurfad, amuriv, amurivr, amurkif, amurmal, amurpuri, amurtimid, amurvivab, amurvivabl, arab, arababil, arabbyvab, arabbyvabl, arabfad, arabiv, arabivr, arabkif, arabmal, arabpuri, arabtimid, arabvivab, arabvivabl, arb, arbabil, arbbyvab, arbbyvabl, arbfad, arbiv, arbivr, arbkif, arbmal, arbpuri, arbr, arbrabil, arbrbyvab, arbrbyvabl, arbrebyvab, arbrebyvabl, arbrefad, arbrekif, arbremal, arbrepuri, arbretimid, arbrevivab, arbrevivabl, arbrfad, arbriv, arbrivr, arbrkif, arbrmal, arbrpuri, arbrtimid, arbrvivab, arbrvivabl, arbtimid, arbvivab, arbvivabl, batir, batirabil, batirbyvab, batirbyvabl, batirfad, batiriv, batirivr,

Genetic algorithm hyperparameters

Parameter	Value	Parameter	Value
Island population	200	Migration interval	30
Number of islands	750	Migration ratio	10%
Total population	150,000	Mutation rate	0.8
Selection method	Rank	Crossover rate	0.2
Total generations	5,000	Rule set crossover	Rule pair crossover
Elite size	5%	HMM crossover	Emissions only

batirkif, batirmal, batirpuri, batirtimid, batirvivab, batirvivabl, burk, burkabil, burkbyvab, burkbyvabl, burkebyvab, burkebyvabl, burkefad, burkekif, burkemal, burkepuri, burketimid, burkevivab, burkevivabl, burkfad, burkiv, burkivr, burkkif, burkmal, burkpuri, burktimid, burkvivab, burkvivabl, byl, bylabil, bylbyvab, bylbyvabl, bylfad, byliv, bylivr, bylkif, bylmal, bylpuri, byltimid, bylvivab, bylvivabl, film, filmabil, filmbyvab, filmbyvabl, filmebyvab, filmebyvabl, filmefad, filmekif, filmemal, filmepuri, filmetimid, filmevivab, filmevivabl, filmfad, filmiv, filmivr, filmkif, filmmal, filmpuri, filmtimid, filmvivab, filmvivabl, filt, filtabil, filtbyvab, filtbyvabl, filtfad, filtiv, filtivr, filtkif, filtmal, filtpuri, filtr, filtrabil, filtrbyvab, filtrbyvabl, filtrebyvab, filtrebyvabl, filtrefad, filtrekif, filtremal, filtrepuri, filtretimid, filtrevivab, filtrevivabl, filtrfad, filtriv, filtrivr, filtrkif, filtrmal, filtrpuri, filtrtimid, filtrvivab, filtrvivabl, filttimid, filtvivab, filtvivabl, furyr, furyrabil, furyrbyvab, furyrbyvabl, furyrfad, furyriv, furyrivr, furyrkif, furyrmal, furyrpuri, furyrtimid, furyrvivab, furyrvivabl, kapt, kaptabil, kaptbyvab, kaptbyvabl, kaptebyvab, kaptebyvabl, kaptefad, kaptekif, kaptemal, kaptepuri, kaptetimid, kaptevivab, kaptevivabl, kaptfad, kaptiv, kaptivr, kaptkif, kaptmal, kaptpuri, kapttimid, kaptvivab, kaptvivabl, karaf, karafabil, karafbyvab, karafbyvabl, karaffad, karafiv, karafivr, karafkif, karafmal, karafpuri, karaftimid, karafvivab, karafvivabl, klad, kladabil, kladbyvab, kladbyvabl, kladfad, kladiv, kladivr, kladkif, kladmal, kladpuri, kladtimid, kladvivab, kladvivabl, klop, klopabil, klopbyvab, klopbyvabl, klopfad, klopiv, klopivr, klopkif, klopmal, kloppuri, kloptimid, klopvivab, klopvivabl, krab, krababil, krabbyvab, krabbyvabl, krabfad, krabiv, krabivr, krabkif, krabmal, krabpuri, krabtimid, krabvivab, krabvivabl, kup, kupabil, kupbyvab, kupbyvabl, kupfad, kupiv, kupivr, kupkif, kupl, kuplabil, kuplbyvab, kuplbyvabl, kuplebyvab, kuplebyvabl, kuplefad, kuplekif, kuplemal, kuplepuri, kupletimid, kuplevivab, kuplevivabl, kuplfad, kupliv, kuplivr, kuplkif, kuplmal, kuplpuri, kupltimid, kuplvivab, kuplvivabl, kupmal, kuppuri, kuptimid, kupvivab, kupvivabl, kurb, kurbabil, kurbbyvab, kurbbyv-

abl, kurbebyvab, kurbebyvabl, kurbefad, kurbekif, kurbemal, kurbepuri, kurbetimid, kurbevivab, kurbevivabl, kurbfad, kurbiv, kurbivr, kurbkif, kurbmal, kurbpuri, kurbtimid, kurbvivab, kurbvivabl, kuverk, kuverkabil, kuverkbyvab, kuverkbyvabl, kuverkfad, kuverkiv, kuverkivr, kuverkkif, kuverkl, kuverklabil, kuverklbyvab, kuverklbyvabl, kuverklebyvab, kuverklebyvabl, kuverklefad, kuverklekif, kuverklemal, kuverklepuri, kuverkletimid, kuverklevivab, kuverklevivabl, kuverklfad, kuverkliv, kuverklivr, kuverklkif, kuverklmal, kuverklpuri, kuverkltimid, kuverklvivab, kuverklvivabl, kuverkmal, kuverkpuri, kuverktimid, kuverkvivab, kuverkvivabl, kylt, kyltabil, kyltbyvab, kyltbyvabl, kyltebyvab, kyltebyvabl, kyltefad, kyltekif, kyltemal, kyltepuri, kyltetimid, kyltevivab, kyltevivabl, kyltfad, kyltiv, kyltivr, kyltkif, kyltmal, kyltpuri, kylttimid, kyltvivab, kyltvivabl, mord, mordabil, mordbyvab, mordbyvabl, mordfad, mordiv, mordivr, mordkif, mordmal, mordpuri, mordr, mordrabil, mordrbyvab, mordrbyvabl, mordrebyvab, mordrebyvabl, mordrefad, mordrekif, mordremal, mordrepuri, mordretimid, mordrevivab, mordrevivabl, mordrfad, mordriv, mordrivr, mordrkif, mordrmal, mordrpuri, mordrtimid, mordrvivab, mordrvivabl, mordtimid, mordvivab, mordvivabl, odor, odorabil, odorbyvab, odorbyvabl, odorfad, odoriv, odorivr, odorkif, odormal, odorpuri, odortimid, odorvivab, odorvivabl, parl, parlabil, parlbyvab, parlbyvabl, parlebyvab, parlebyvabl, parlefad, parlekif, parlemal, parlepuri, parletimid, parlevivab, parlevivabl, parlfad, parliv, parlivr, parlkif, parlmal, parlpuri, parltimid, parlvivab, parlvivabl, provok, provokabil, provokbyvab, provokbyvabl, provokfad, provokiv, provokivr, provokkif, provokmal, provokpuri, provoktimid, provokvivab, provokvivabl, prut, prutabil, prutbyvab, prutbyvabl, prutfad, prutiv, prutivr, prutkif, prutmal, prutpuri, pruttimid, prutvivab, prutvivabl, purp, purpabil, purpbyvab, purpbyvabl, purpfad, purpiv, purpivr, purpkif, purpmal, purppuri, purpr, purprabil, purprbyvab, purprbyvabl, purprebyvab, purprebyvabl, purprefad, purprekif, purpremal, purprepuri, purpretimid, purprevivab, purprevivabl, purprfad, purpriv, purprivr, purprkif, purprmal, pur-

prpuri, purprtimid, purprvivab, purprvivabl, purptimid, purpvivab, purpvivabl, romp, rompabil, rompbyvab, rompbyvabl, rompfad, rompiv, rompivr, rompkif, rompmal, romppuri, rompr, romprabil, romprbyvab, romprbyvabl, romprebyvab, romprebyvabl, romprefad, romprekif, rompremal, romprepuri, rompretimid, romprevivab, romprevivabl, romprfad, rompriv, romprivr, romprkif, romprmal, romprpuri, romprtimid, romprvivab, romprvivabl, romptimid, rompvivab, rompvivabl, tab, tababil, tabbyvab, tabbyvabl, tabfad, tabiv, tabivr, tabkif, tabl, tablabil, tablbyvab, tablbyvabl, tablebyvab, tablebyvabl, tablefad, tablekif, tablemal, tablepuri, tabletimid, tablevivab, tablevivabl, tablfad, tabliv, tablivr, tablkif, tablmal, tablpuri, tabltimid, tablvivab, tablvivabl, tabmal, tabpuri, tabtimid, tabvivab, tabvivabl, yrl, yrlabil, yrlbyvab, yrlbyvabl, yrlebyvab, yrlebyvabl, yrlefad, yrlekif, yrlemal, yrlepuri, yrletimid, yrlevivab, yrlevivabl, yrlfad, yrliv, yrlivr, yrlkif, yrlmal, yrlpuri, yrltimid, yrlvivab, yrlvivab.

Bibliography

Adamidis, Panagiotis. 1994. Review of parallel genetic algorithms bibliography. *Aristotle Univ. Thessaloniki, Thessaloniki, Greece, Tech. Rep* .

Azencott, Robert. 1992. *Simulated annealing: parallelization techniques*, volume 27, chapter 4,5,6. Wiley-Interscience.

Baker, James Edward. 1985. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their applications*, 101–111. Hillsdale, New Jersey.

Bennett III, Forrest H, John R Koza, James Shipman, and Oscar Stiffelman. 1999. Building a parallel computer system for $18,000 that performs a half peta-flop per day. In *Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2*, 1484–1490. Morgan Kaufmann Publishers Inc.

Berwick, Robert C. 1985. *The acquisition of syntactic knowledge*. Cambridge, Massachusetts: MIT Press.

Cantú-Paz, Erick. 1998. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis* 10:141–171.

Chaitin, Gregory J. 1966. On the length of programs for computing finite binary

sequences. *Journal of the ACM* 13:547–569.

Chater, Nick. 1999. The search for simplicity: A fundamental cognitive principle? *The Quarterly Journal of Experimental Psychology: Section A* 52:273–302.

Chater, Nick, and Paul Vitányi. 2007. 'Ideal learning' of natural language: Positive results about learning from positive evidence. *Journal of Mathematical Psychology* 51:135–163.

Chau, Chak-Wai, Sam Kwong, CK Diu, and Wolfgang R Fahrner. 1997. Optimization of hmm by a genetic algorithm. In *Acoustics, Speech, and Signal Processing, 1997. ICASSP-97., 1997 IEEE International Conference on*, volume 3, 1727–1730. IEEE.

Chomsky, Noam. 1965. *Aspects of the theory of syntax*. Cambridge, MA: MIT Press.

Chomsky, Noam, and Morris Halle. 1968. *The sound pattern of English*. New York: Harper and Row Publishers.

De Jong, Kenneth Alan. 1975. Analysis of the behavior of a class of genetic adaptive systems. Doctoral Dissertation.

Dell, François. 1981. On the learnability of optional phonological rules. *Linguistic Inquiry* 12:31–37.

Fortin, Félix-Antoine, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research* 13:2171–2175.

Goldberg, David E. 1989. *Genetic algorithms in search, optimization and machine learning*. Addison Wesley.

Goldberg, David E., and Kalyanmoy Deb. 1991. A comparative analysis of selection schemes used in genetic algorithms. In *Foundations of genetic algorithms*, volume 1, 69–93. Elsevier.

Gordon, V Scott, and Darrell Whitley. 1993. Serial and parallel genetic algorithms as function optimizers. In *ICGA*, 177–183.

Greening, Daniel R. 1990. Parallel simulated annealing techniques. *Physica D: Nonlinear Phenomena* 42:293–306.

Holland, John H. 1975. Adaptation in natural and artificial systems. an introductory analysis with application to biology, control, and artificial intelligence. *Ann Arbor, MI: University of Michigan Press* 439–444.

Kaplan, Ronald M., and Martin Kay. 1994. Regular models of phonological rule systems. *Computational Linguistics* 20:331–378.

Kirkpatrick, Scott, C. Daniel Gelatt, and Mario P. Vecchi. 1983. Optimization by simulated annealing. *Science* 220:671–680.

Kolmogorov, Andrei Nikolaevic. 1965. Three approaches to the quantitative definition of information. *Problems of Information Transmission (Problemy Peredachi Informatsii)* 1:1–7.

Li, Ming, and Paul Vitányi. 2008. *An introduction to Kolmogorov complexity and its applications*. Berlin: Springer Verlag, 3rd edition.

Marcus, Gary F. 1993. Negative evidence in language acquisition. *Cognition* 46:53–85.

Mitchell, Melanie. 1998a. *An introduction to genetic algorithms*. MIT press.

Mitchell, Melanie. 1998b. *An introduction to genetic algorithms*, chapter 1.9.

MIT press.

Onbaşoğlu, Esin, and Linet Özdamar. 2001. Parallel simulated annealing algorithms in global optimization. *Journal of Global Optimization* 19:27–50.

Pearce, David J. 2005. An improved algorithm for finding the strongly connected components of a directed graph. *Victoria Univ., Wellington, NZ, Tech. Rep* .

Rasin, Ezer, Iddo Berger, and Roni Katzir. 2015. Learning rule-based morpho-phonology. *http://ling.auf.net/lingbuzz/002800/* .

Rasin, Ezer, Iddo Berger, Nur Lan, and Roni Katzir. 2018. Learning rule-based morpho-phonology. Ms., MIT and Tel Aviv University.

Rasin, Ezer, and Roni Katzir. 2016. On evaluation metrics in Optimality Theory. *Linguistic Inquiry* 47:235–282.

Razali, Noraini Mohd, John Geraghty, et al. 2011. Genetic algorithm performance with different selection strategies in solving tsp. In *Proceedings of the world congress on engineering*, volume 2, 1134–1139. International Association of Engineers Hong Kong.

Rissanen, Jorma. 1978. Modeling by shortest data description. *Automatica* 14:465–471.

Solomonoff, Ray J. 1964. A formal theory of inductive inference, parts I and II. *Information and Control* 7:1–22, 224–254.

Stanley, Kenneth O, and Risto Miikkulainen. 2002. Evolving neural networks through augmenting topologies. *Evolutionary computation* 10:99–127.

Tattersall, Ian. 2012. *Masters of the planet: The search for our human origins*. Macmillan.

Wexler, Kenneth, and Rita M. Manzini. 1987. Parameters and learnability in binding theory. In *Parameter setting*, ed. Thomas Roeper and Edwin Williams, 41–76. Dordrecht, The Netherlands: D. Reidel Publishing Company.

Whitley, Darrell, Soraya Rana, and Robert B Heckendorn. 1999. The island model genetic algorithm: On separability, population size and convergence. *Journal of Computing and Information Technology* 7:33–47.

Won, Kyoung-Jae, Thomas Hamelryck, Adam Prügel-Bennett, and Anders Krogh. 2007. An evolutionary method for learning hmm structure: prediction of protein secondary structure. *BMC bioinformatics* 8:357.

Won, Kyoung-Jae, Adam Prügel-Bennett, and Anders Krogh. 2004. Training hmm structure with genetic algorithm for biological sequence analysis. *Bioinformatics* 20:3613–3619.

Won, Kyoung-Jae, Adam Prugel-Bennett, and Anders Krogh. 2006. Evolving the structure of hidden markov models. *IEEE Transactions on Evolutionary Computation* 10:39–49.

Yada, Tetsushi, Masato Ishikawa, Hidetoshi Tanaka, and Kiyoshi Asai. 1994. Dna sequence analysis using hidden markov model and genetic algorithm. *Genome Informatics* 5:178–179.

Zomaya, Albert Y, and Rick Kazman. 2010. Simulated annealing techniques. In *Algorithms and theory of computation handbook*, 33.1–33.18. Chapman & Hall/CRC.