Learning Morpho-phonology Using a Genetic Algorithm

Nur Lan
Tel Aviv University
nurlan@mail.tau.ac.il

Ezer Rasin
Leipzig University
ezer.rasin@uni-leipzig.de

Roni Katzir
Tel Aviv University
rkatzir@tauex.tau.ac.il

ABSTRACT

As part of language acquisition, children acquire knowledge of the sound pattern of their language – their knowledge of morpho-phonology – which goes well beyond the plain phonetic form of words. According to a long-standing model in linguistics, the child acquires this knowledge by searching a complex and possibly infinite hypothesis space consisting of all learnable morpho-phonological grammars. We present a general learning algorithm for morpho-phonology that uses the principle of Minimum Description Length to choose between competing hypotheses and that relies on a genetic algorithm to search the hypothesis space. We describe the genetic algorithm and its evolutionary operators which are defined over linguistic representations. The genetic algorithm performs better than an earlier search algorithm used for learning in this domain and allows the learner to successfully acquire complex morpho-phonological processes in natural language.

CCS CONCEPTS

• **Computing methodologies** → **Genetic algorithms**; *Phonology / morphology*;

KEYWORDS

Linguistics, Language Acquisition, Minimum Description Length

ACM Reference format:
Nur Lan, Ezer Rasin, and Roni Katzir. 2019. Learning Morpho-phonology Using a Genetic Algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference 2019, Prague, Czech Republic, July 13–17, 2019 (GECCO '19),* 9 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

As part of language acquisition, children acquire knowledge of the sound pattern of their language – their knowledge of morpho-phonology – which goes well beyond the plain phonetic form of words. For example, English speakers know that the word [dɔgz] 'dogs' is formed by combining the noun [dɔg] 'dog' with the plural suffix [-z].[1] English speakers do not simply memorize the singular and plural forms of every word. Instead, they generalize beyond their input and are able to generate plural forms for words they have never heard before, such as the plural [wʌgz] 'wugs' for the nonce word [wʌg] 'wug' [5]. The task of separating unsegmented words in the input into morphemes (and learning the ways in which morphemes can be combined) is the *segmentation task*.

The segmentation task for the child is non-trivial, given that their input does not contain overt information signaling the boundaries between different morphemes. It is further complicated by the fact that phonological processes can obscure the pronunciation of morphemes. For example, while the English plural suffix is pronounced as [-z] in [dɔgz] 'dogs', it is pronounced as [-s] in [kæts] 'cats'. The pronunciation of the plural morpheme in different contexts is systematic: it is pronounced as the voiceless consonant [-s] after voiceless consonants (like [t]) but as the voiced consonant [-z] elsewhere.[2] Here, again, speakers do not memorize the pronunciation of the plural morpheme together with the noun that it combines with. Rather, they choose the appropriate variant ([-s] or [-z]) depending on the final consonant of the noun, even when they have never heard the plural form of the noun before [5]. The task of learning the systematic distribution of sounds during language acquisition is the *phonological task*. In this work, we present an unsupervised learning algorithm that relies on a genetic algorithm to address both learning challenges – the segmentation task and the phonological task – simultaneously.

A plausible learning algorithm for morpho-phonology will need to be able to generalize beyond the input, since speakers know that nonce forms like [wʌgz] are grammatical (they obey the rules of the language). At the same time, however, a learning algorithm must not over-generalize, since speakers know that nonce forms like *[kætz] are ungrammatical (they do not obey the rules of the language).[3] This knowledge is acquired in the absence of direct negative evidence that forms like *[kætz] are ungrammatical (see [7, 29] for a discussion of negative evidence in language acquisition). That is, the learner must avoid making over-generalizations without receiving restrictive feedback, and at the same time avoid overfitting the data in order to be able to generalize.

Our approach to the learning challenge is based on the idea of simplicity: grammars are considered good by the learner if they lead to an overall short description of the input data. Using this criterion, formalized through the principle of Minimum Description Length (MDL; [35]) and described in Section 2, the learner can search through the space of possible grammars and try to find the one that minimizes the overall description length of the input data. As we review below, MDL addresses the need to generalize at an intermediate level, avoiding both over- and under-generalization. This has made MDL a useful approach to various learning tasks in the domain of language (see [6, 9, 12, 15, 16, 18, 23, 36, 38], among others), including aspects of morpho-phonological learning (see

[1] Square brackets indicate the surface phonological representation of words, as opposed to, e.g., their orthographic representation.

GECCO '19, July 13–17, 2019, Prague, Czech Republic

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[2] Phonetically, a voiced consonant is produced with a vibration in the vocal folds, while a voiceless consonant is not.

[3] An asterisk indicates an ungrammatical form.

[19, 20, 31, 33]). However, while the MDL metric itself appears to be a promising guide for choosing between competing grammars, using it to search through the (potentially infinite) hypothesis space of possible grammars has proven difficult. One approach in the literature has been to simplify this search, typically by focusing first on segmentation and only later on phonology, and restricting the phonological part to morpheme boundaries (see, e.g., [19, 20, 31]). This, however, risks limiting the scope of the learner to a proper subset of what children can acquire (for example, in phenomena in which there are dependencies between the phonological and segmentation tasks; see [34]). A different approach has been taken in [33], where segmentation and phonology are acquired simultaneously. However, that attempt was limited by the search procedure that it relied on – namely, simulated annealing – to extremely simple patterns and very small datasets. We believe that genetic algorithms, due to their inherent parallelizability, offer a way to search through the complex hypothesis space of joint segmentation and phonology, thus making it possible to overcome the limitation of earlier learners in the literature. Our goal in this paper is to describe the design and implementation of such a genetic algorithm, focusing on the genetic operators that allow it to navigate the complex search space of morpho-phonological grammars.

The paper is organized as follows. Section 2 introduces MDL learning for morpho-phonology. Section 3 describes the implementation details of the proposed genetic algorithm. Section 4 presents sample simulation results. Alternative search methods and implementation choices are discussed in Section 5. Section 6 concludes.

2 A SIMPLICITY CRITERION FOR MORPHO-PHONOLOGY

In the context of morpho-phonological learning, a hypothesis is a grammar – a characterization of speakers' knowledge. According to a long-standing model in phonology [11], a morpho-phonological grammar consists of two components: a lexicon, which stores the morphemes of the language along with information on how they can be combined; and phonological rewrite rules, which convert stored lexical entries into their surface phonological forms. In our toy English example from before, and considering the words [dɔgz] and [kæts], the grammar could consist of a lexicon with three morphemes {/dɔg/, /kæt/, /-z/} along with the information that /-z/ optionally follows /dɔg/ and /kæt/, and a phonological rule that converts /-z/ to [-s] after a voiceless consonant.[4] The phonological rule will derive the correct pronunciation of the plural morpheme in different contexts.

For the purpose of evaluating competing grammars, the MDL principle states the following; when choosing between grammars, given the observed data D, choose the grammar G that minimizes the sum:

$$|G| + |D{:}G| \tag{1}$$

[4]Forward slashes (//) indicate the phonological representation of a lexical entry (as opposed to its orthographic representation or its surface phonological representation).

In morpho-phonology, the two summands in (1) have the following meaning:

$$\underbrace{\underbrace{0110100100100000001101}_{\text{Lexicon}}\underbrace{10101010010}_{\text{Phonological Rules}}}_{G}\underbrace{01100001011001000110100 1}_{D\,:\,G}$$

- $|G|$ – the number of bits used to encode the grammar G, which contains a lexicon and a set of phonological rules.
- $|D{:}G|$ – the number of bits used to encode the words in D ('data'), given that the grammar G has been learned.

By minimizing the two factors in (1), the learner balances between an over-generalizing grammar and an over-restrictive one. Simpler grammars are often more permissive; if G becomes more permissive, $|G|$ will tend to become shorter, while $|D{:}G|$ will grow, since specifying the observed data from among more alternative surface forms that G can generate will require additional bits. On the other hand, if G becomes more restrictive (for example, if it stores surface forms in the lexicon without segmentation), $|D{:}G|$ will diminish, but $|G|$ will be larger. This balancing between grammar simplicity and restrictiveness makes it possible to generalize while avoiding over-generalization, and to store a lexicon without overfitting the data.

2.1 Representations

In order to apply the MDL principle to learning morpho-phonology, it is necessary to measure the encoding length of the hypotheses that make up the learner's search space. For this we need to explicitly state the representation and encoding scheme for our hypotheses. The search algorithm will operate directly over the chosen representations.

2.1.1 Phonological rules. A standard assumption in phonology is that each segment (consonant or vowel) is represented as a set of binary features that describe its articulatory properties. For example, the first segment in 'dog', [d], can be represented using the features '+cons' (which indicates that the segment is consonantal), '+voice' (which indicates that the segment is voiced), the features '+coronal' and '-velar' (which provide information about the place of articulation of the consonant), and so on:

$$\begin{bmatrix} +cons \\ +voice \\ +coronal \\ -velar \\ \cdots \end{bmatrix} \tag{2}$$

The representation of segments as sets of binary features is motivated by evidence that language speakers can make phonological generalizations over sub-segmental features [24, 25].

The segments and features that are available to the learner are illustrated in the simplified Table 1. The actual table used in our simulations is significantly larger and would ideally contain all options available to the child learner.

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

Table 1: Partial phonological feature table

Phonological rules are represented in the rewrite-rule format from Chomsky & Halle's *The Sound Pattern of English* (SPE; [11]), depicted in Figure 1.

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

Figure 1: SPE rewrite rule format

A and B are feature vectors like the one in (2) that may also be empty ($\emptyset$). X and Y are (possibly empty) sequences of feature vectors that specify the left and right context in which the rule is allowed to apply. Informally, a rule that follows this format is interpreted as follows: if a segment whose features match the ones in A appears between segments that match features X on the left and Y on the right, then it is modified so that its features match the ones in B. *optional* is a boolean flag specifying whether the rule applies optionally or obligatorily.

The rule in (3) is one possible statement of the English voicing assimilation rule mentioned in Section 1. The rule states that non-sonorant consonants (like /z/ but unlike /n/) become voiceless when preceded by any voiceless segment:

$$\begin{bmatrix} +cons \\ -sonorant \end{bmatrix} \rightarrow [-voice] \,/\, [-voice] \_\_ \tag{3}$$

In order to measure the binary encoding length of a rule, it must first be serialized into a binary string. This is done using a conversion table that maps rule components to binary strings. The length of each symbol in bits is determined by the overall number of components in the rule. A naive binary encoding is used: if there are n possible symbols in a rule, each binary string will be of length $\lceil \log_2 n \rceil$ bits.

2.1.2 Lexicon. The lexicon is represented as a Hidden Markov Model (HMM). Each state has a transition function, in addition to a set of emissions. Here, the emissions in each state correspond to morphemes, and transitions represent legal morpheme combinations. Each path from the initial state of the HMM to its final state corresponds to a sequence of morphemes that produces a valid word in the language. An example HMM for the English words /kæt/ and /dɔg/ and the optional suffix /-z/ is given in Figure 2. This HMM can produce both the singular and plural forms of each noun (/dɔg/, /dɔgz/, /kæt/, /kætz/), and these will be subject to the application of phonological rules (which would convert, for example, /kætz/ into [kæts]).

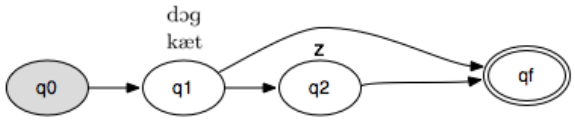


Figure 2: HMM representing a limited English lexicon with the plural suffix /-z/

In order to encode the lexicon as a binary string and measure its length, the HMM is serialized similarly to the phonological rules above. Each symbol in the final encoding is a binary string of length $\lceil \log_2 n \rceil$ bits, where n is the number of symbols required to represent an HMM: all states, segments (that make up emissions), and delimiters that make it possible to reconstruct the HMM from its flattened string form.

2.2 Data given the grammar

In order to measure the encoding length of the data given the grammar – the $|D{:}G|$ term in the MDL sum – we need to consider how the data are generated by the grammar. To describe every data point (a surface phonological form) given the grammar, we need to specify the following choices. First, the choice of a combination of morphemes from the lexicon (which corresponds to a specification of a path from the HMM's initial state to its final state). Second, we need to specify the application of optional phonological rules (obligatory rules always apply and do not require additional specification). Brute-force calculation of the smallest number of bits required for the description of every data point is generally infeasible, since a grammar typically generates many output forms, often infinitely many. To make the calculation efficient, we compile the lexicon and rules into a finite-state transducer (FST) using the method described in [27] and use dynamic programming for parsing. The outputs of the FST corresponds to all words generated by the grammar. In order to calculate $|D{:}G|$, each word in D is fed through the FST, and the number of bits required to encode each transition choice along the way is taken as its encoding length. Once again a naive binary encoding is used, in which a selection between n transitions has a uniform probability and costs $\lceil \log_2 n \rceil$ bits. Note that a grammar may generate more or fewer outputs than the target grammar: for example, if the grammar is over-generalizing, it will generate more words than D contains; in this case the MDL criterion will assign a longer description length to $|D{:}G|$ because more encoding choices will be made along the way.

Figure 3 is an example FST for a grammar consisting of the English voicing assimilation rule in (3) and a lexicon with the morphemes /dɔg/ and /kæt/ and the plural suffix /-z/.

Let us follow a sample calculation of the encoding length of the surface form [kæts] – the result of combining /kæt/ with the plural suffix /-z/ and then applying the voicing assimilation rule. First, the transition from q_0 to q_1 is deterministic and costs 0 bits. Then, each of the two outgoing transitions from q_1 costs $\lceil \log_2 2 \rceil = 1$ bits,

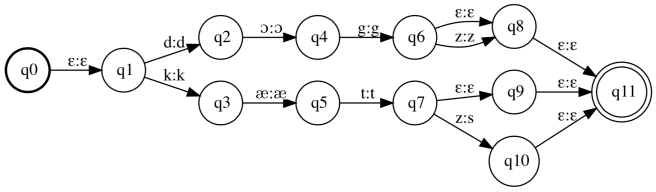


Figure 3: FST for a toy English grammar with a voicing assimilation rule and a lexicon containing /dɔg/ **and** /kæt/ **and the plural suffix** /-z/

representing the selection of one of the two morphemes. We take the transition to q_3 representing /kæt/. The next non-deterministic choice occurs at the transition from q_7 to either q_9 or q_{10}, which represents specifying whether the plural suffix should be added and costs $\lceil \log_2 2 \rceil = 1$ bit. Note that since the correct environment for the rule (a consonant following a voiceless consonant) applies here, the transducer translates the voiced /z/ to the voiceless /s/ when transitioning from q_7 to $q_1 0$; this is in contrast to the transition from q_6 to q_8 where the plural suffix remains voiced when following a voiced consonant. The next transition to the final accepting state is deterministic and costs 0 bits. The final encoding length for [kæts] is thus 2 bits. This process is repeated for each surface form in the corpus and summed to form $|D{:}G|$.

3 GENETIC ALGORITHM

The learner is exposed only to phonetic transcriptions of utterences, and is expected to induce the correct underlying rules and lexicon that generated them. This is done by searching for a hypothesis with a minimal description length. Since the hypothesis space is infinite, brute-force search for the optimal hypothesis is infeasible. Moreover, the hypotheses that the learner is manipulating are discrete objects which do not easily convert to continuous representations, and we are not aware of any differentiable approximations to the MDL target function. At present this makes the task irrelevant to gradient-based optimizers like artificial neural networks.

Genetic Algorithms (GAs; [26]) are a general optimization strategy that can traverse complicated search spaces and can operate on discrete symbolic objects, like the ones manipulated by this learner. Working in the context of phonological rules and morphological lexicons, the basic operators used by GAs – mutation and crossover – needed to be implemented in a way that will be expressive enough to allow the algorithm to develop hypotheses effectively.

The following subsections describe the implementation details of the GA. We wish first to emphasize that the search algorithm in this work is chosen solely for its ability to efficiently optimize the present learning task; we assume nothing about the way the human brain actually performs the search, of which we still know very little.[5] The code for the learner is available at: github.com/REDACTED.

[5]It is worth noting that works such as [13] and [44] have drawn parallels between language acquisition and Darwinian evolution.

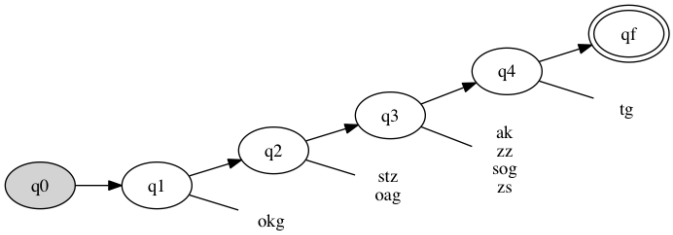


Figure 4: Random HMM

$$R_1 : [+cons] \rightarrow [+coronal] / \_\_\_ [+voice] \, (optional)$$
$$R_2 : [+liquid] \rightarrow \emptyset / [+coronal, +strident] \_\_\_ \quad (obligatory)$$

Figure 5: Random rule set

HMM	Rules
Add random emission	Add random rule
Remove emission	Remove rule
Concatenate two emissions	Demote rule
Copy emission to another state	Promote rule
Add segment to emission	Flip 'optional' flag value
Replace segment in emission	Add random feature set
Add new state	Remove a feature set
Remove existing state	Add feature to a set
Add transition	Remove feature from set
Remove transition	Flip feature binary value

Table 2: Available mutations

3.1 Population initialization

Upon simulation start, a population of size N is generated randomly. Each hypothesis in the population consists of an HMM representing the lexicon, and a rule set.

3.1.1 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 (Figure 4). Each state is assigned random emissions, using the table of available segments. The maximum number of states, maximum number of emissions per state, and their maximum length are configurable parameters of the simulation.

3.1.2 Random rules. A random set of rules is generated, each containing a random feature set for each of the target, change, and context positions (Figure 5). The maximum number of rules in a set and the maximum number of features per position are configurable parameters of the simulation.

3.2 Mutation

An individual undergoes mutation using one of the mutations in Table 2. One of the hypothesis' components – lexicon or rule set – is selected with uniform probability (0.5), and one mutation is selected from the relevant list, also with uniform probability.

3.3 Crossover

The crossover operator was implemented separately for the lexicon component and the rule component. Two hypotheses are mated by applying crossover to one of the components, selected at random with uniform probability (0.5).

3.3.1 Lexicon crossover. One naive way to implement crossover between HMMs is to represent both parent HMMs as transition matrices, and then to crossover the rows and their respective emissions. After switching the rows, the offspring HMMs can be reconstructed from the result transition matrices (see Figure 6). This crossover implementation is in the spirit of the canonical GA binary-string crossover, in the sense that it flattens the HMM structure to binary matrices similar to bit-strings and then applies random crossover to the flat representation.

However, HMMs contain more layers of information – emissions and state transitions – and flattening this structure may produce defective offspring. For example, in the context of morphological segmentation, the HMM's functionality often arises from a combination of states, e.g., two consecutive states that represent stems and suffixes. If the rows are switched randomly, the states may get separated and the hypothesis will become invalid. Moreover, we have noticed that during an intermediate stage of the learning process, an HMM which is not yet optimal often contains a loop of transitions, which allows it to freely concatenate segments to create all possible morphemes; this is not an optimal hypothesis because it is wasteful in terms of $|D{:}G|$, but it may still evolve into hypotheses that can better represent the corpus. If this loop is broken the HMM will fail to generate all morphemes and the hypothesis will become invalid. In addition, the learned segmentation may be idiosyncratic to the current overall grammar, where it interacts with the phonological rules.

We have implemented several HMM crossover operators that manipulate both emissions and transitions, described in Section 5.1. We opted eventually for a simple crossover implementation that moves only the emissions between parents while keeping transitions intact. The rationale is that an accepting parent, who had hopefully learned a somewhat-correct segmentation, may benefit from morphemes it still has not discovered. Emissions from a state in one parent move with uniform probability to the respective state in the other parent. This crossover implementation leaves the evolution of the HMM topology to the mutation operator alone.

3.3.2 Rules crossover. Similarly to lexicons, crossover for rules needed to be implemented in a way that will preserve useful rules that were evolved. In our implementation a rule is crossed-over with the other parent's rule at the same position, with uniform probability (0.5). If one parent has more rules, the smaller set can receive rules from the other, and the overall number of rules stays intact. As with lexicon crossover, this implementation leaves the evolution of rules mostly to mutation.

3.4 Fitness

The fitness of an individual is set to the hypothesis' encoding length (number of bits in $|G|+|D:G|$), as described in Section 2.2. However, for GA implementations that operate on non-continuous hypothesis spaces, like this learner, it is common to design the fitness function so that invalid hypotheses also receive a fitness score, usually with a penalty, to help guide the search towards meaningful hypotheses [30]. Thus, in our implementation fitness scores are also assigned to hypotheses that cannot generate the data, according to the following heuristic. Note that we are dealing with a minimization task, where lower fitness is better.

Hypothesis Fitness Calculation.

- A valid hypothesis' fitness value is taken as $|G| + |D : G|$
- An invalid hypothesis that can't parse all words gets a penalized fitness value equal to:

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

- *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, while ordering 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 do not represent any data.

3.5 Parallelization

One naive way to parallelize the canonical genetic algorithm is to distribute the mutation, crossover, and evaluation operations across multiple processes, in a 'worker pool' structure in which each worker process receives a hypothesis, applies one of the operations to it, and sends the result back to the master process. Since each application of the basic genetic operators is independent of other individuals' results, this parallelization scheme can potentially yield a speedup linear in the number of processes. However, it also incurs an overhead cost due to intensive communication between processes, which may cancel out the parallelization gain.

We adopt instead the Island Model parallelization scheme [1, 8, 21]. Inspired by evolutionary speciation and niching, the model divides the overall population into 'islands'. Each island is in essence an independent vanilla genetic algorithm, which evolves a subpopulation. Each island is run in a parallel process that is mostly independent of other islands. Once in every number of generations a migration step occurs, during which some individuals from each island are copied to an adjacent island. After each island has completed the overall preconfigured number of generations, the best hypothesis from all islands is taken as the optimal solution.

Unlike the naive parallelization scheme, here the processes rarely communicate, and when they do, only a small fraction of the population is sent, which improves run times significantly.

The rate with which the migration step occurs is called the migration interval, and the percentage of individuals that migrate is called the migration ratio (M_{ratio}). We implemented two migration

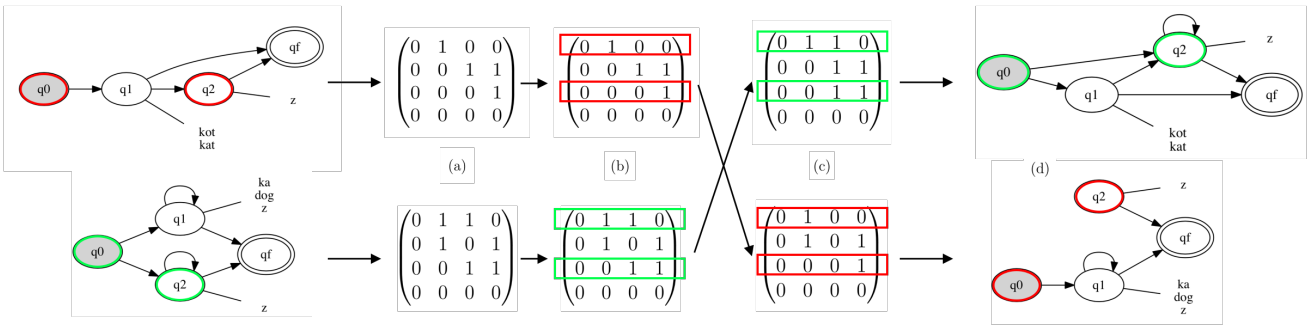


Figure 6: Naive HMM crossover using transition matrices. (a) The transitions of each parent HMMs are represented as transition matrices. (b) Rows 0 and 2 are selected at random for crossover. (c) New transition matrices are created by switching the selected rows and their emissions. (d) The offspring HMMs are constructed from the new transition matrices.

topologies. The first is a naive circular scheme that always sends individuals from one island to its immediate neighbor, 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 can be slow in propagating novelties between populations: a feature discovered in one island will take many generations to propagate to distant islands. It is also prone to bottlenecks that may be caused by island processes that are slower or have crashed. We opted instead for a topology that also arranges the islands in a ring, but performs migration in a round-robin fashion [39]. This topology enables genetic innovation to spread faster between islands and helps to avoid bottlenecks.

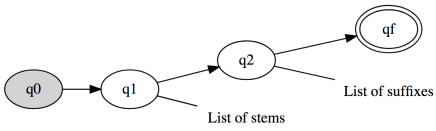
For selecting outgoing migrants and assimilating incoming ones, we adopt the method in [39]: the fittest M_{ratio} individuals of an island are sent as migrants, and the same amount of the worst individuals are replaced. For island synchronization, we follow the recommendations in [4], mainly: the islands never wait for each other (*asynchronous migration*), since some islands may run more slowly 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 migrants from different islands before its migration step, the most recent group of migrants is taken. These heuristics help speed up the search, since almost no synchronization delays occur and the dependency between islands is minimized.

3.6 Selection

We use rank-based selection [3], a variant of fitness-proportionate roulette-wheel selection [17], which assigns 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 individuals with lower fitness to survive, thus maintaining population diversity. In addition, we employ Elitism – a simple yet effective mechanism that prevents losing the best individuals in a population [14]: a configurable percentage of the best individuals in each generation is kept for the next generation, independent of mutation and crossover.

stem\suffix	∅	-z	-ing	-er	⋯
rent	rent	rents	renting	renter	
klaimb	klaimb	klaimbz	klaimbing	klaimber	
kros	kros	krosiz	krosing	kroser	
analaiz	analaiz	analaiziz	analaizing	analaizer	
⋯					

Table 3: English simulation sample data

$G =$ {
RULES:
$R_1 : \emptyset \rightarrow [+high, -back] / [+strident]$ ___ $[+strident]$
$R_2 : [-son] \rightarrow [-voice] / [-voice]$ ___
LEXICON:


Stems = rent, klaimb, glu, komit, straik, drim, kontrol, . . .
Suffixes = ϵ, z, ing, er, al, abl, ment, i, . . .
}

Description length: $|G| + |D{:}G| = 894.1 + 19{,}914.4 = 20{,}808.5$

Figure 7: Final grammar for the English rule-ordering simulation. It includes the correct epenthesis and voicing assimilation rules, in correct order, and a segmented lexicon.

4 RESULTS: ENGLISH SEGMENTATION AND RULE ORDERING

The dataset for our simulation was generated by an artificial grammar modeled after the interaction of voicing assimilation and epenthesis in English. As mentioned before, English voicing assimilation de-voices consonants like /z/ when preceded by a voiceless consonant. Another rule of epenthesis inserts the vowel [ɪ] between two sibilant consonants, as in [glæsɪz] 'glasses'. In the rule-based

model of phonology we have adopted in this paper, the interaction between multiple rules is modeled using rule ordering. To derive forms such as [glæsɪz], where voicing assimilation does not apply and the suffix remains voiced, the epenthesis rule is ordered before the voicing-assimilation rule. When epenthesis applies to the morpheme combination /glæs-z/, it disrupts the adjacency between the suffix and the preceding consonant, rendering assimilation inapplicable. The opposite ordering would have derived the incorrect form *[glæsɪs]. To learn the English pattern correctly, the learner needs to acquire the two rules, as well as the correct ordering (epenthesis before assimilation), jointly with segmentation.

The learner was presented with 250 words generated by combining 25 stems and 10 suffixes and applying the two rules in order. A sample of the data is provided in Table 3. The simulation used 750 islands with a population of 200 (a total population of 150,000), a crossover rate of 0.2 and a mutation rate of 0.8, and ran for 5,000 generations.[6] The learner converged on the expected lexicon and on the two rules – epenthesis (R_1) and assimilation (R_2) – and their correct ordering (Figure 7).

5 ALTERNATIVES

5.1 Alternative HMM Crossover Operators

HMMs are made of two layers of information – emissions and transitions. In the segmentation task they represent the morphemes and ways of combining them, which are highly co-dependent. Randomly switching these two layers may render the HMM useless in terms of generating the data. The crossover operation thus needs to be conservative enough so as not to destroy valuable learned aspects of an HMM, but at the same time expressive enough to result in the inheritance of valuable traits.

We experimented with crossover implementations inspired by two main types of previous works: those designed specifically for HMMs, and those designed for formalisms resembling HMMs, mainly directed graphs. [40–43] have used genetic algorithms for learning HMM topologies for DNA sequence analysis, using various graph representations similar to the ones below. [10] used a transition-matrix representation similar to the one described in Section 3.3.1. Since HMMs can be seen as a specific case of directed graphs, we also considered implementations used for evolving graphs, including implementations designed to evolve efficient neural-network topologies [37] (which are usually represented as directed weighted graphs).

In what follows, we describe crossover operators that we developed that were not sufficient to evolve successful hypotheses without the help of mutation. These operators improved convergence rates compared to simulations in which only mutation was allowed, but the model still heavily relies on mutations. Mutations turned out important for evolving HMM emissions, as well as the internal parts of rules, which were not able to be acquired through crossover alone. Simulations in which only crossover was allowed converged prematurely on bad solutions. We leave the development of alternative, more successful crossover operators for future work.

[6]The simulation ran on three AWS c5.18xlarge machines with 72 vCPUs each (3.0 GHz Intel Xeon), each machine running 250 islands.

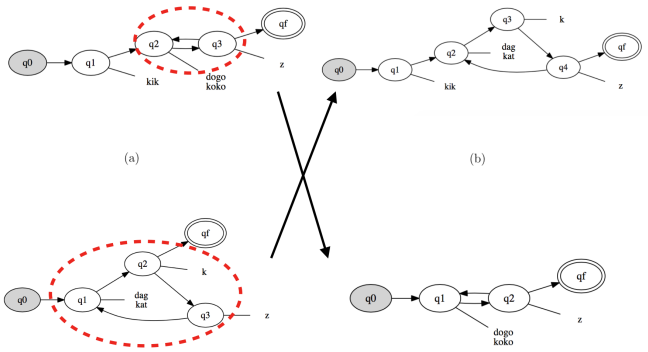


Figure 8: HMM crossover using connected components. (a) A connected component is selected from each parent. (b) Each offspring receives a component from the other parent. Note how loops are preserved.

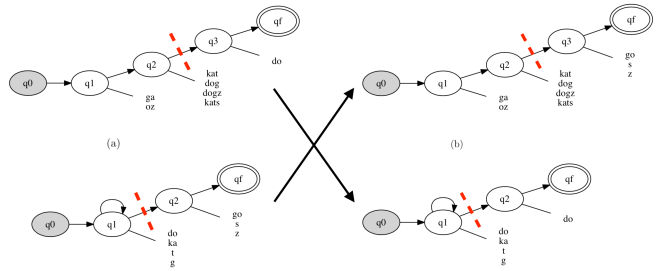


Figure 9: HMM crossover using subgraphs. (a) A transition arc in each parent is selected as a cutoff point – for the upper parent the arc from q_2 to q_3 and for the lower parent from q_1 to q_2. (b) The offspring are constructed by crossing-over the parent graphs above and below the cutoff points.

5.1.1 Connected components crossover (Figure 8). Each parent HMM is considered as a directed graph, and its connected components are found. Two connected components are randomly chosen from each parent and switch places. One incoming arc and one outgoing arc from each component are kept and re-attached to the new inserted component. The emissions of each crossed-over state move with their respective states. The motivation for this crossover operator is to switch groups of consecutive states instead of single, randomly-selected states; since the learner often needs to evolve meaningful consecutive transitions, e.g., stems and suffixes, this operator may help preserve these structures. It also prevents loops from being broken, which may help preserve valid hypotheses.

5.1.2 Subgraph crossover (Figure 9). This implementation is based on the crossover operator used in Genetic Programming (GP). In GP, programs are often represented as tree graphs, in which each node contains a functional operator. The common crossover operator for 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 cutoff point. This implementation respects the learned functionality of the tree and has the potential of importing functionality from both parents to create superior offspring.

An arc in each parent HMM is chosen randomly. The HMM is cut at the chosen arc, and the subgraph stemming from it is switched with the subgraph from the other parent. Since HMMs are not necessarily trees and may contain cycles, in our implementation only arcs that serve as incoming/outgoing arcs of a connected component are selected as cutoff places. Emissions move with their respective states.

5.2 Comparison with alternative search methods

Alternative methods for searching highly complex hypothesis spaces include Markov Chain Monte Carlo (MCMC) and Variational Bayesian methods, which make it possible to navigate the space by sampling. Simulated Annealing (SA; [28]) was used in [33] as an optimization method for a morpho-phonological learner. SA is designed to reach a global optimum in complex hypothesis spaces and can operate on discrete objects. As a close relative of MCMC random walk methods, SA performs a random walk over the hypothesis space following a simple hypothesis selection heuristic and a cooling agenda.

While [33] obtained positive results with SA, simulations took days to converge even on small corpora. In this section we note that run times improve significantly with GA compared to SA on small corpora. In addition, GA succeeds on larger corpora, whereas no comparable results are currently available with SA.

While the canonical version of SA is sequential, parallelization methods for SA have been proposed [2, 22, 32, 45]. To compare SA with our parallelized version of GA, we implemented two common parallelization methods for SA: the periodically-interacting scheme, where several SA 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 annealing chain is run, with each decision step consisting of choosing between p neighbor hypotheses instead of 1, making use of p parallel processes. These methods did not lead to qualitatively better run times with SA. This can be attributed to the fact that these methods require sending hypothesis data back-and-forth between processes for evaluation; by profiling the application, we observed that the overhead of communication between processes had offset the run time which may have been gained by parallelization.

Table 4 provides a summary of the comparison between GA and SA and reports simulation run times using identical corpora and hardware.[7] The displayed times are times that successful simulations took to complete. Note that this is not a robust comparison since it involves different algorithms with different stop conditions: SA stops when temperature reaches a bottom threshold (starting at 75 here) while GA stops after a configured number of generations (10,000 here). We believe, however, that the big differences provide a good indication of the performance gain with GA.

The use of GA did not only speed up existing simulations, but also made it possible to run successful simulations on larger and more natural corpora than before. The English corpus presented in Section 4 contains 250 real English words with 17 segments made of 11 phonological features. Earlier results obtained with SA on the same morpho-phonological pattern used 8 segments and 5 features and were limited to nonce words and small corpus sizes (32-105 words). The English simulation presented in Section 4 took 12 hours to converge using GA, while the SA learner did not show any indication of convergence after twice the amount of time for the same corpus. We attribute this advance mostly to GA's population model compared to SA's single hypothesis model, and to the massive parallelization possible in GA compared to SA's serial mode of operation.

[7] Intel Xeon 3.30GHz with 16GB RAM. The GA simulations use multiple processors.

Corpus	Words	SA	GA
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

Table 4: Time to successful simulation completion

6 SUMMARY

This paper presented the design and implementation of an unsupervised learning algorithm for morpho-phonology that relies on a genetic algorithm to search the complex space of joint segmentation and phonology. The use of a genetic algorithm allowed the learner to improve on a previous proposal for learning segmentation and phonology jointly, which used different search algorithms and was limited to extremely simple patterns and very small datasets. We compared the design choices of our learner with several conceivable alternatives that did not perform as well in terms of navigating the morpho-phonological hypothesis space (e.g., alternative crossover operators over HMMs).

ACKNOWLEDGMENTS

This work was supported by the Microsoft Azure and the Amazon Web Services research programs.

REFERENCES

[1] Panagiotis Adamidis. 1994. Review of parallel genetic algorithms bibliography. *Aristotle Univ. Thessaloniki, Thessaloniki, Greece, Tech. Rep* (1994).
[2] Robert Azencott. 1992. *Simulated annealing: parallelization techniques.* Vol. 27. Wiley-Interscience, Chapter 4,5,6.
[3] James Edward Baker. 1985. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their applications.* Hillsdale, New Jersey, 101–111.
[4] Forrest H Bennett III, 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.* Morgan Kaufmann Publishers Inc., 1484–1490.
[5] Jean Berko. 1958. The child's learning of English morphology. *Word* 14, 2-3 (1958), 150–177.
[6] Michael Brent. 1999. An Efficient, Probabilistically Sound Algorithm for Segmentation and Word Discovery. *Computational Linguistics* 34, 1–3 (1999), 71–105.
[7] Roger Brown. 1970. Derivational complexity and order of acquisition in child speech. *Cognition and the development of language* (1970).
[8] Erick Cantú-Paz. 1998. A survey of parallel genetic algorithms. *Calculateurs paralleles, reseaux et systems repartis* 10, 2 (1998), 141–171.
[9] Nick Chater and Paul Vitányi. 2007. 'Ideal Learning' of Natural Language: Positive Results about Learning from Positive Evidence. *Journal of Mathematical Psychology* 51 (2007), 135–163.

[10] Chak-Wai Chau, 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*, Vol. 3. IEEE, 1727–1730.

[11] Noam Chomsky and Morris Halle. 1968. *The Sound Pattern of English*. Harper and Row Publishers, New York.

[12] Alexander Clark. 2001. *Unsupervised Language Acquisition: Theory and Practice*. Ph.D. Dissertation. University of Sussex.

[13] R. Clark. 1992. The selection of syntactic knowledge. *Language Acquisition* 2, 2 (1992), 83–149.

[14] Kenneth Alan De Jong. 1975. *Analysis of the behavior of a class of genetic adaptive systems*. Ph.D. Dissertation.

[15] Carl de Marcken. 1996. *Unsupervised Language Acquisition*. Ph.D. Dissertation. MIT, Cambridge, MA.

[16] Timothy Mark Ellison. 1994. *The machine learning of phonological structure*. Ph.D. Dissertation. University of Western Australia.

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

[18] John Goldsmith. 2001. Unsupervised Learning of the Morphology of a Natural Language. *Computational Linguistics* 27, 2 (2001), 153–198.

[19] John Goldsmith. 2006. An Algorithm for the Unsupervised Learning of Morphology. *Natural Language Engineering* 12, 3 (2006), 1–19.

[20] Sharon Goldwater and Mark Johnson. 2004. Priors in Bayesian Learning of Phonological Rules. In *7th Annual Meeting of the ACL Special Interest Group on Computational Phonology*. 35–42.

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

[22] Daniel R Greening. 1990. Parallel simulated annealing techniques. *Physica D: Nonlinear Phenomena* 42, 1-3 (1990), 293–306.

[23] Peter Grünwald. 1996. A Minimum Description Length Approach to Grammar Inference. In *Connectionist, Statistical and Symbolic Approaches to Learning for Natural Language Processing*, G. S. S. Wermter and E. Riloff (Eds.). Springer, 203–216.

[24] Morris Halle. 1962. Phonology in generative grammar. *Word* 18, 1-3 (1962), 54–72.

[25] Morris Halle. 1978. Knowledge unlearned and untaught: What speakers know about the sounds of their language. (1978).

[26] John H Holland. 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* (1975), 439–444.

[27] Ronald M. Kaplan and Martin Kay. 1994. Regular Models of Phonological Rule Systems. *Computational Linguistics* 20, 3 (1994), 331–378.

[28] Scott Kirkpatrick, C. Daniel Gelatt, and Mario P. Vecchi. 1983. Optimization by Simulated Annealing. *Science* 220, 4598 (1983), 671–680.

[29] Gary F Marcus. 1993. Negative evidence in language acquisition. *Cognition* 46, 1 (1993), 53–85.

[30] Melanie Mitchell. 1998. *An introduction to genetic algorithms*. MIT press, Chapter 1.9.

[31] Jason Naradowsky and Sharon Goldwater. 2009. Improving Morphology Induction by Learning Spelling Rules. In *IJCAI*. 1531–1536.

[32] Esin Onbaşoğlu and Linet Özdamar. 2001. Parallel simulated annealing algorithms in global optimization. *Journal of Global Optimization* 19, 1 (2001), 27–50.

[33] Ezer Rasin, Iddo Berger, and Roni Katzir. 2015. Learning rule-based morphophonology. *http://ling.auf.net/lingbuzz/002800/* (2015).

[34] Ezer Rasin, Nur Lan, and Roni Katzir. 2019. Simultaneous learning of vowel harmony and segmentation. (2019). Poster to be presented at SCiL 2019.

[35] Jorma Rissanen. 1978. Modeling by Shortest Data Description. *Automatica* 14 (1978), 465–471.

[36] Jorma Rissanen and Eric Sven Ristad. 1994. Language Acquisition in the MDL Framework. In *Language computations: DIMACS Workshop on Human Language, March 20-22, 1992*. Amer Mathematical Society, 149.

[37] Kenneth O. Stanley, Jeff Clune, Joel Lehman, and Risto Miikkulainen. 2019. Designing neural networks through neuroevolution. *Nature Machine Intelligence* 1, 1 (2019), 24–35. https://doi.org/10.1038/s42256-018-0006-z

[38] Andreas Stolcke. 1994. *Bayesian Learning of Probabilistic Language Models*. Ph.D. Dissertation. University of California at Berkeley, Berkeley, California.

[39] Darrell Whitley, 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, 1 (1999), 33–47.

[40] Kyoung-Jae Won, Thomas Hamelryck, Adam Prügel-Bennett, and Anders Krogh. 2007. An evolutionary method for learning HMM structure: prediction of protein secondary structure. *BMC bioinformatics* 8, 1 (2007), 357.

[41] Kyoung-Jae Won, Adam Prügel-Bennett, and Anders Krogh. 2004. Training HMM structure with genetic algorithm for biological sequence analysis. *Bioinformatics* 20, 18 (2004), 3613–3619.

[42] Kyoung-Jae Won, Adam Prugel-Bennett, and Anders Krogh. 2006. Evolving the structure of hidden Markov models. *IEEE Transactions on Evolutionary Computation* 10, 1 (2006), 39–49.

[43] Tetsushi Yada, Masato Ishikawa, Hidetoshi Tanaka, and Kiyoshi Asai. 1994. DNA sequence analysis using hidden Markov model and genetic algorithm. *Genome Informatics* 5 (1994), 178–179.

[44] Charles D Yang. 2002. *Knowledge and learning in natural language*. Oxford University Press on Demand.

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