Some Learning Algorithms for Phonotactics

Daniel M. Albro

June 13, 2000

1 The Problem – Phonotactic Learning

1.1 Background

One of the first tasks children face in the process of acquiring their first languages is to develop a characterization of which sound sequences are legal or illegal in their native languages. Knowledge of this characterization—phonotactics—develops at or before 10 months of age (Jusczyk *et al.* 1994).

Knowledge they (seem to) have: ability to recognize their own language, knowledge of its surface segment inventory.

Knowledge they don't (seem to) have: morphology, semantics, syntax.

1.2 Goal

- Learning of phonotactic information expressed by Optimality Theoretic constraint rankings
- An algorithm that is
  - robust in the face of noisy data or variation
  - capable of learning constraint rankings from surface-only data

1.3 Existing Algorithms

Previous algorithms can be put into three categories:

1. *Not robust, not suited for phonotactic*s: Tesar's earlier Constraint Demotion algorithms (Tesar & Smolensky 1993; Tesar 1997) fail (become overly permissive, crash, or fail to terminate) in the presence of noise, and are not designed to deal with the phonotactic learning situation, where underlying forms are unknown (they tend to overgeneralize due to the absence of negative evidence).

2. *Robust, but not suited for phonotactic*s: Boersma's Gradient Learning Algorithm (Boersma 1997).

3. *Not robust, but suited for phonotactic*s: The algorithms of Prince and Tesar (1999) and of Hayes (1999) can learn phonotactics (some comparisons to my algorithm later), but fail in the presence of noise.

1.4 The EDBPR Algorithm—Robust and Suited for Phonotactics

The Error-Driven Bayesian Phonotactic Ranking (EDBPR) Algorithm is implemented on top of the author's UCOTP (Albro 1998) implementation of Primitive Optimality Theory (Eisner 1997). It uses Bayesian reasoning to deal with noisy data/variation.

2 Exemplification

2.1 Of Phonotactic Learning

Comments on phonotactic learners in general:

- Given: valid surface forms recast as underlying forms

- Produce: a ranking that allows as outputs of generation the smallest possible superset of the forms encountered during learning. This ranking takes URs similar to the forms encountered during learning and produces unchanged surface forms, but changes other URs into forms similar to those encountered during learning.

- Such a ranking has these characteristics: Markedness constraints are ranked as high as possible and faithfulness constraints as low as possible. Given this, when choosing among faithfulness constraints to promote above markedness constraints, it is better to promote constraints that are specific than to promote generally applicable faithfulness constraints. This is because faithfulness constraints have the effect of allowing (some) URs to pass through unchanged. Specific faithfulness constraints apply to fewer URs, and thus promoting them provides less of a chance to overgeneralize.

2.1.1 The azba problem

- From Prince & Tesar (1999), an example of how their constraint specificity metric (and that of Hayes (1999)) is inadequate for learning the most conservative ranking in some cases.
- Concerns a fictional language similar in some respects to Greek and Russian. Voicing is distinctive in stops, but not in fricatives, and voiced fricatives are only found in regressive assimilations.
- Problem: find a grammar that doesn't allow voiced fricatives in too many places.
- Example words:
  - [pa], [ba]
  - [ap], [ab]
  - [sa], *[za]
  - [as], *[az]
  - [apsa], *[abza], *[apza], *[absa]
  - [aspa], [azba], *[asba], *[azpa]
- Constraints (in the most restrictive ranking, their terminology):

M:AGREE(voi)	Adjacent obstruents agree in voicing
M:*b	Stops must be voiceless
M:*z	Fricatives must be voiceless
F:STOP-VOI/ONS	Preserve stop voicing in surface onsets
F:STOP-VOI	Preserve stop voicing
F:FR-VOI/Ons	Preserve fricative voicing in surface onsets
F:FR-VOI	Preserve fricative voicing

- Learning is error-driven, so it requires input of forms the strict grammar rejects. These are [ba], [ab], and [azba].
  - /ba/→[pa] is evidence of F:STOP-VOI **or** F:STOP-VOI/ONS ≫ M:*b
  - /ab/→[ap] is evidence of F:STOP-VOI ≫ M:*b
  - /azba/→[aspa] is evidence of F:STOP-VOI **or** F:STOP-VOI/ONS **or** F:FR-VOI ≫ M:*b **and** M:*z
- Tesar and Prince's basic algorithm (Biased Constraint Demotion), which ignores constraint specificity, chooses F:STOP-VOI in all instances of choice here. Such a choice allows [abza], so it's too permissive a grammar.
- Because of this, they modify their algorithm to preferentially promote faithfulness constraints that are more specific. They define *specific* as follows (my words): *Constraint C_1 is more specific than C_2 if C_1 applies to a strict subset of the candidates C_2 applies to.* However, given this definition, the algorithm can't distinguish between F:STOP-VOI/ONS ≫ M:*z and F:FR-VOI ≫ M:*z, since neither faithfulness constraint accepts a proper subset of the candidates accepted by the other. However, F:FR-VOI ≫ M:*z is too permissive—it allows [z] in all positions.
- It turns out that the UCOTP implementation provides a straightforward way to determine a numerical rating of how much of the possible overall candidate set (that is, the set of all possible words in any language) a constraint might apply to. OTP represents phonological structures in terms of gestural scores, and, essentially, it is possible to

determine for any OTP constraint how many different types of vertical slices (time-slices) within a gestural score the constraint might affect. This specificity metric corresponds more directly to the idea that rerankings should keep as small a grammar as possible—a further improvement, not yet made, but, I think, possible, would measure specificity on the candidate set available at the point in the ranking where a constraint is to be placed.

- In the metric used here F:STOP-VOI/ONS is more specific than F:FR-VOI, and the algorithm presented here thus correctly chooses it, since it can be shown to apply to a smaller number of time slices, and thus leads to more restrictive grammars.

2.2 Of Robustness in the Face of Noise and/or Variation

2.2.1 An example from Hayes (1999): Pseudo-Korean

- A subset of Korean in which the only permissible surface segments are [d], [t], [t^h], and [a].
- Of these, [d] only appears in intervocalic position, and [t^h] never appears word-finally.
- This state of affairs is accounted for by virtue of the following constraints (Hayes' terminology):

*[-SON, +VOICE]	Obstruents are voiceless by default
*[+VOICE][-VOICE][+VOICE]	*Voiceless segments between voiced ones
*[+SPREAD GLOTTIS]	*Aspiration
*[+VOICE, +SPREAD GLOTTIS]	Voicing is incompatible with aspiration
IDENT(ASP)/___V	Preserve underlying aspiration before vowels
IDENT(ASP)	Preserve underlying aspiration
IDENT(VOICE)/___V	Preserve underlying voicing before vowels
IDENT(VOICE)	Preserve underlying voicing

2.2.2 Hayes' algorithm

- Given a diet of correct surface forms, Hayes' algorithm comes up with the following ranking strata:

*[+VOICE, +SPREAD GLOTTIS]	1
IDENT(ASP)/___V	2
*[+VOICE][-VOICE][+VOICE] *[+SPREAD GLOTTIS]	3
*[-SON, +VOICE]	4
IDENT(ASP) IDENT(VOICE)/___V IDENT(VOICE)	5

(Table 1)

- However, when given illegal inputs such as [data] or [at^{h}], even once in a data set consisting of millions of words, Hayes' algorithm will produce a grammar that promotes IDENT(ASP) above *[+SPREAD GLOTTIS], and IDENT(VOICE)/___V above *[+VOICE][-VOICE][+VOICE] and *[-SON, +VOICE].

2.2.3 What the EDBPR Algorithm does

1. Estimate likelihood of getting evidence for all possible relaxations of the strictest possible grammar (expressed as pairwise ranking probabilities $P(C_1 \gg C_2)$) by running Monte Carlo trials with randomly-ranked input grammars. That is, we generate random rankings and random URs, run the URs through the rankings and find the strictest grammars that accept the resulting surface forms, keeping track of how often particular relaxations $C_1 \gg C_2$ occur. This simulates trying to learn all possible phonotactic systems and thus seeing what evidence for rerankings from the strictest grammars is likely in general.

2. For each input word, determine the strictest grammar that accepts it and note which relaxations were necessary.

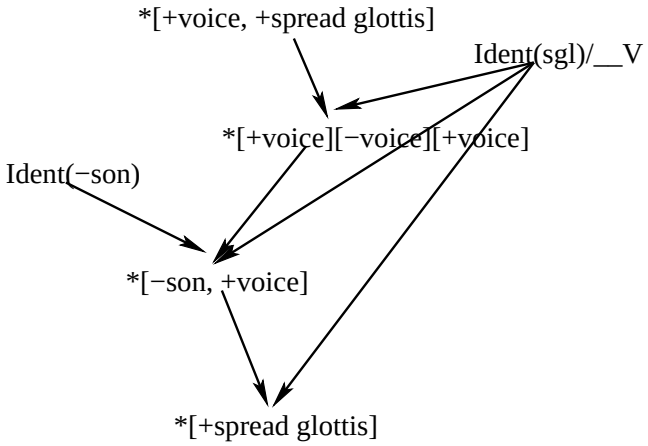
3. Using the likelihoods from (1) and the number of times each pairwise reranking was necessary in (2), calculate the probability of each pairwise reranking.

2.2.4 What results from this?

- If the Monte Carlo trials show that across all possible phonotactic systems evidence for a particular relaxation (ranking pair) is rare, then the learner needs to encounter only a few examples in their own language that force that relaxation before the learner decides the relaxation is necessary, since crosslinguistically rare words are unlikely to appear in linguistic noise.
- If a particular relaxation is evidenced quite frequently during the random trials, then it is also deemed likely to show up in noise, so it takes a large number of exemplifications before the learner decides it is a necessary ranking in its language.

2.2.5 Output in the case of Pseudo-Korean

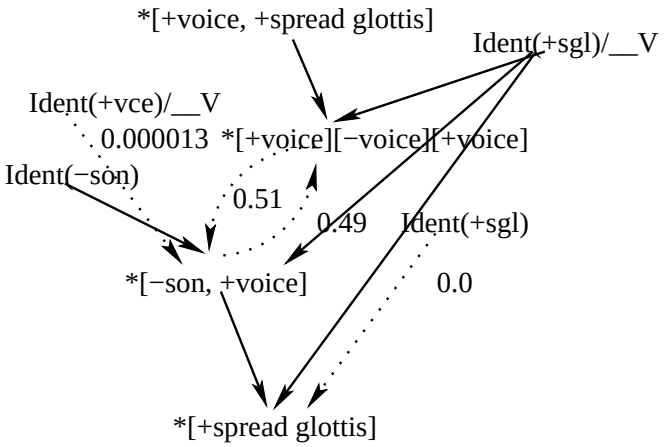
Given 200 random valid Pseudo-Korean surface forms, the EDBPR algorithm came up with the rankings shown in figure 1 (each with 100% certainty):



(Figure 1)

These are essentially consistent with the output of Hayes' algorithm. Given those same words, but with the illegal form [data] thrown in, the learner adds the ranking pairs IDENT(VOICE)/__V ≫ *[-SON, +VOICE] and *[-SON, +VOICE] ≫ *[+VOICE][-VOICE][+VOICE]. As it happens, in the Monte

Carlo trials the learner discovers that the first ranking pair is often evidenced, but the second is not. At the end, the first ranking pair is given a likelihood of 0.00013 and the second a likelihood of 0.98. Since the likelihood of *[+VOICE][-VOICE][+VOICE] ≫ *[-SON, +VOICE] was calculated to be 1.0, the probabilities are scaled to 0.49 and 0.51, respectively. If the form [at^{h}] is also given, the learner adds the ranking pair IDENT(+SPREAD GLOTTIS) ≫ *[+SPREAD GLOTTIS], which is frequently enough evidenced in the Monte Carlo trials that it is given a likelihood of 0. The resulting ranking diagram is thus as follows (figure 2):



(Figure 2)

With this output, the learner would be expected to find [ata] fairly acceptable (in speech it would be in free variation with [ada] as a realization of /ata/), [da] extremely marginally acceptable, and [at^{h}] not acceptable at all.

A The EDBPR Algorithm

The basic goal of the algorithm is to determine the probability for each single-pair constraint ranking $A \gg B$ that the ranking is necessary in the grammar accepting the smallest possible subset of the input forms. (In the current implementation these rankings are conceived of as being applied atop a set of base strata in which markedness constraints outrank faithfulness constraints, and faithfulness constraints are ranked such that the least permissive constraints are on top).

A.1 Overall:

1. ("Parameter Estimation"): Learn how to learn phonotactics (find out the likelihood of getting evidence for various ranking pairs, given randomized input grammars).
2. ("Ranking Reinforcement"): Input surface forms and reinforce pairwise rankings found necessary to accept those surface forms.
3. ("Calculation of $P(H|O)$"): Calculate the probability of necessity for each possible pairwise ranking, given the observed inputs from step 2.
4. Return to step 1, if necessary.

A.2 Parameter Estimation:

Repeat ad nauseam:

1. Generate a random input.
2. Generate a random ranking (Keep track for each pair C_1, C_2 of the number of estimation trials with ranking $C_1 \gg C_2$ vs. $C_2 \gg C_1$.)
3. Generate an output by running the input through UCOTP (Albro 1998) with the given ranking.
4. Create the mirror image of that output as an input.
5. Determine a minimally permissive set of pairwise rankings that produces just the expected output.
6. Reinforce (by incrementing a counter) the individual $C_1 \gg C_2$ pairs in the set of rankings produced.

When a minimal number of trials of each pairwise ranking has been run, calculate the Bayesian Estimation parameters as follows:

$$p_{H|H} = \frac{r_H^e}{n_H^e}$$

$$p_H = \frac{r_H^e}{n^e}$$

(H represents a pairwise ranking hypothesis, *e.g.* that $C_1 \gg C_2$ is necessary, $p_{H|H}$ is the probability that the $C_1 \gg C_2$ is found to be necessary by the algorithm when confronted with a random output from a grammar in which $C_1 \gg C_2$ holds, p_H is the probability that the algorithm finds $C_1 \gg C_2$ necessary regardless of whether $C_1 \gg C_2$ holds in the input grammar, r_H^e is the number of estimation reinforcements of H, n_H^e is the number of estimation trials in which ranking H holds in the input grammar, and n^e is the total number of estimation trials).

A.3 Ranking Reinforcement

When the algorithm has performed enough estimation and receives actual surface forms, it follows steps 4–6 of the estimation algorithm for each surface form, thus gathering a value r_H for each pairwise ranking possibility.

A.4 Calculation of $P(H|O)$

Probability of necessity of a pairwise ranking (call this H) given an observed number of reinforcements is calculated via Bayes' Theorem:

$$P(H|O) = \frac{P(O|H)P(H)}{P(O)} = \frac{P(O|H)P(H)}{P(O|H)P(H) + P(O|\overline{H})P(\overline{H})}$$

where $P(H)$ is initially 0.5 (it could be continuously adjusted by Bayesian updating to ease computation, if necessary);

$$P(O|H) = \left(\begin{array}{c} n \\ m \end{array} \right) p^m (1-p)^{n-m}$$

(it's a binomial distribution), where p is the value $p_{H|H}$ calculated during estimation, n is the number of observed inputs, and m is the number of reinforcements of the hypothesis (r_H); and

$$P(O|\overline{H}) = \left(\begin{array}{c} n \\ m \end{array} \right) \overline{p}^m (1-\overline{p})^{n-m}$$

where $\overline{p} = p_n p_H$, that is, the expected probability of noise (I've used 0.1) p_n times the probability in general of reinforcement of this hypothesis .

Note that the binomial terms $\left(\begin{array}{c} n \\ m \end{array} \right)$ cancel out, a fact which makes these equations much more feasible to compute.

A.5 Determination of Strictest Rerankings from a Basic Ranking

1. Take the base ranking (markedness constraints on top, other constraints sorted in order of increasing permissiveness (number of segment time slices allowed by the constraint)) and generate an output from the UR mirror image of the input SR.

2. Use the standard tableau method to compare violations of the expected vs. actual output. This gives a set of reranking pairs which is generally more permissive than necessary

   $$(C_1 \text{ or } C_2 \text{ or } \ldots \gg C_4 \text{ and } C_5 \text{ and } \ldots)$$

3. Apply all of the rerankings to the current ranking. Then generate a new output, and return to step 2, unless the expected output was generated.

4. Sort the resultant rankings $C_i \gg C_j$ by the desirability of C_i as a top ranker (prefer markedness constraints over faithfulness constraints, prefer strict faithfulness constraints over permissive faithfulness constraints).

5. Moving in this order, from the least to the most acceptable rerankings, try to remove each ranking $C_i \gg C_j$ from the set of necessary rankings. This is possible if applying the remaining rankings to the base ranking still produces the expected output (and only the expected output). This step can introduce new rankings, in which case we return to step 3.

B Some Proposed Modifications to EDBPR

B.1 A Faster Strictest Rankings Algorithm

The algorithm presented above for finding the strictest rankings that accept some input or set of inputs is perhaps not as efficient as it might be. It works by first running an overgenerating version of Tesar's Error-Driven Constraint Demotion algorithm, which involves running the UCOTP output generation algorithm repeatedly until the proper output is obtained, and then eliminating constraint rankings one by one by running output generation for each

one. Output generation can be slow, so it would be better to minimize this. Proposed here is a Biased Recursive Constraint Demotion algorithm based on a Linear-time Recursive Constraint Demotion algorithm gleaned from an anonymous review paper. That algorithm works as follows:

1. For each losing candidate, create a disjunction of constraints for which the winning candidate does better. While building these, maintain for each constraint a list of the disjunctions that mention it, and for each disjunction maintain a list of the constraints that disjunction outranks (constraints for which the losing candidates do better). For each constraint, also maintain the number of disjunctions that outrank it.

2. Make a list *undom* of all constraints with 0 outranking disjunctions.

3. Set *stratum* $\leftarrow 0$

4. Until all constraints are ranked

   (a) Remove the first constraint in the list *undom*, and call it C_1. If there is no such constraint, fail.

   (b) For each disjunct that mentions C_1

      i. For each constraint C_o outranked by the disjunct

         A. Reduce the number of disjunctions outranking C_o by one.

         B. If this falls to 0, add C_o to *undom*

      ii. (this disjunct may now be considered inactive)

   (c) Set stratum of C_1 to *stratum*

   (d) Increment *stratum*

This algorithm is by itself not suitable for this problem, since it will tend to put the faithfulness constraints on top, but biasing it is fairly simple. The only thing that needs to be done is to keep the *undom* list sorted in the following way:

1. Rank the constraints from least to most permissive (use the specificity metric above, or another one to be presented). Keep track of this ranking.

2. To assign an order in *undom* between two constraints C_1 and C_2:

(a) If C_1 has some active disjuncts that mention it and C_2 doesn't, put C_1 first (and vice versa).

(b) Otherwise, order C_1 and C_2 according to the permissiveness ranking assigned in step (1).

The strictest-ranking algorithm then works by taking assigning the strictest possible ranking initially, seeing what outputs are produced by the input using that ranking, and if non-optimal outputs are produced, using them as candidates in the Biased RCD described above. The output of Biased RCD is a new ranking. Apply the ranking, produce outputs, and continue in this way until the expected output (and only that output) is achieved. This algorithm is no longer a linear RCD, of course, but is more on the order of $O(n^2 \log n)$.

B.2 A slightly different permissiveness metric

(This one is technical (and vague), and might be skipped). A possible method for determining permissiveness is to go directly with the idea that a constraint is strict if it allows few identity outputs to go through, and lax otherwise. Basically, we can produce as an input an FSM that represents a slice of all possible identity candidates, regardless of input (the length of this slice should be just long enough to allow all constraint WFSAs to be exercised). This slice represents a finite number of candidates. Constraints can then be transformed such that no state is marked initial, and markedness constraints are then judged in terms of the number of identity candidates they allow through. Faithfulness constraints must be further transformed from correspondence constraints to anti-correspondence constraints, and then can be measured against each other in the same way that markedness constraints were.

C A Method for Learning Constraints, Given Non-Noisy Data

If we look at the Biased RCD algorithm, we can see that as it works it builds up a knowledge-base about how the constraints must be outranked. That is, for each constraint, we accumulate a set of disjunctions of constraints that must outrank it. If we assume noisy data, we can't fully trust this knowledge,

because any given input that contributed to it might not have been valid. If we instead focus on a situation in which noise is not present (or assume that some filtering device is possible), we can use this knowledge to get a more powerful learning algorithm for phonotactics. Presumably, this knowledge will remain useful as well (in exactly its current form) when moving on past the phonotactic stage, whereas the knowledge embodied in simple ranking pairs, as in the algorithm above might not continue to be as useful.

If we assume the constant build-up of a knowledge-base as described above, we can lay out an algorithm that, in addition to determining phonotactic constraint rankings, also can create new phonotactic constraints (or faithfulness constraints) and adds them to the system. Here it is:

1. Start with a constraint set that is capable (in some ranking) of passing through all identity constraints (*i.e.,* it should have the basic Max, Dep, and Ident constraints, at least).

2. Compute the strictest possible ranking that accepts the inputs. (If this algorithm is to be conceived as iterative, this would be "some limited prefix of the input sequence"). This creates a database of conjuncts $((a \vee b \vee c) \gg d) \wedge ((a \vee b) \gg e)$, etc.

3. Compute a desirability value for this grammar, according to the following metric: take the initial set of "identity candidates" described in section B.2 and pass it through the grammar (with the constraints modified to have no initial state), then count the number of candidates that made it out.

4. Consider possible next steps:

   (a) Create a new constraint from some set of constraint templates and add it to the top (if a phonotactic) or bottom (if a faithfulness constraint) of the grammar. We can keep our knowledge base correct in the presence of this new constraint by adding the new constraint to each disjunct.

   (b) Remove a constraint that is not mentioned in any disjunct. This will not affect the knowledge base.

5. For each move we can compute the desirability value for the resulting grammar as before (possibly after processing some data with it to firm the ranking).

6. This defines a search space, with a neighborhood of grammars around each grammar. We can search this space using whatever strategy we like—possibly hill-climbing or simulated annealing.

D An MDL-like Method for Learning Phonotactic Grammars in the Presence of Noise

Contemplation of the previous algorithm leads to a possible method for dealing with noise, using a Minimal-Description Length-type approach, rather than using Bayesian estimation. The algorithm is basically the same as the previous one, except that:

1. We don't keep an expanding knowledge-base about constraint rankings. Instead, we keep track of all of the inputs.

2. The metric is as follows: desirability(G) = code-length(G) + $\sum_{\text{word}}$ badness(word), where the "badness" of a word is something like 0.9(accepted(word))+ 0.1(perturbation-distance(SG(word), G)), where *accepted* returns 1 if a word is accepted by the grammar, and 0 otherwise, *SG(word)* returns the strictest ranking of the constraints in G that accepts *word*, and the perturbation distance between two grammars is a string-edit distance between the rankings, where the string edit operation is a ranking switch between two adjacently-ranked constraints.

3. Possible grammar modifications (*i.e.,* the definition of the neighbors of a grammar) are as follows:

   (a) Add a constraint to the grammar, as before.

   (b) Remove a constraint from the grammar (this should be restricted to constraints that weren't in the initial constraint set).

   (c) Swap the ranking of two adjacently-ranking constraints.

Modification of the metric parameters should allow variation of the algorithm's sensitivity to noise.

References

Albro, Daniel M., 1998. Evaluation, implementation, and extension of Primitive Optimality Theory. Master's thesis, UCLA.

Boersma, Paul, 1997. How we learn variation, optionality, and probability. [ROA #221].

Eisner, Jason. 1997. Efficient generation in primitive Optimality Theory. In *Proceedings of the ACL*.

Hayes, Bruce P., 1999. Phonological aquisition in optimality theory: the early stages. [Rutgers Optimality Archive #327].

Jusczyk, Peter W., Paul A. Luce, & Jan Charles-Luce. 1994. Infants' sensitivity to phonotactic patterns in the native language. *Journal of Memory and Language* 630–645.

Prince, Alan, & Bruce Tesar, 1999. Learning phonotactic distributions. [Rutgers Optimality Archive #353].

Tesar, Bruce, 1997. Multi-recursive constraint demotion. [Rutgers Optimality Archive #197].

——, & Paul Smolensky. 1993. The learnability of Optimality Theory: an algorithm and some basic complexity results. Technical report, Department of Computer Science and Institute of Cognitive Science, University of Colorado, Boulder. [Rutgers Optimality Archive #2].