Child Phonology Analyzer: processing and analyzing transcribed speech

Chen Gafni

Bar-Ilan University
chen.gafni@gmail.com

ABSTRACT

This paper describes two algorithms for analyzing transcribed speech corpora: (1) identification of phonological processes, and (2) phonological queries. The algorithms are implemented in Visual Basic for Applications for Microsoft Excel, thus exploiting Excel's mass-calculation capabilities to analyze large corpora quickly. The user interface features a set of editable tables that contain definitions of phonological entities. This inclusion provides great flexibility, and allows users to maintain their own working conventions.

Keywords: corpus analysis, phonological processes, language acquisition, phonetic alignment, similarity

1. INTRODUCTION

Automatic analysis of speech corpora is an essential part of the study of complex phenomena, such as phonological development. The introduction of systems such as *Phon* ([6], [7]) has contributed greatly to our ability to detect developmental patterns in child speech. In this paper, I present a new system for analyzing transcribed speech corpora. The algorithms described here include automatic identification of phonological processes, and phonological queries. They are combined with a user-friendly interface in a Microsoft Excel workbook. This implementation allows the exploitation of Excel's built-in mass-calculation capabilities to analyze large corpora quickly. Among other things, the algorithms benefit from the single-step application of formulas to large data sets, and the use of filters to reduce the size of data sets to be analyzed.

The user interface features several editable tables that constitute the phonological knowledge base required by the algorithms. Incorporating these tables in the user interface allows users to maintain their own working conventions, such as the set of relevant phonetic symbols and phonological features, diacritical operations, and definitions of phonological processes.

The algorithms operate on Excel workbooks that contain pairs of (manually transcribed) target-output tokens in adjacent columns (this arrangement can be done by another procedure included in the system).

2. IDENTIFICATION OF PHONOLOGICAL PROCESSES

The proposed algorithm attempts to mimic human analysis of string similarity by dividing the process into a stage of global pattern recognition followed by analysis of local changes. Accordingly, for a given pair of target and output tokens, detection of phonological processes is based on two principles: **(1) maximal identity-mapping**, and **(2) minimized weighted differences**.

2.1. Maximal identity-mapping

This step involves finding the largest sub-string of the output token that is contained in the target token. Initially, Excel's *FIND* function is applied to the entire data range to identify all output tokens that are faithful to their targets, up to (and including) deletions at the edges (e.g., /kæt/ 'cat'→ [kæ]). Changing the argument order in the *FIND* function allows the detection of insertions at word edges.

If the output string is not contained in the target string (and vice versa), the algorithm splits the output to two sub-strings, and iterates through them: in the left branch it removes the rightmost character, and searches the string in the target string from the beginning. In the right branch, the leftmost character is removed, and the search moves from the end of the target string backwards. The iterations continue until both branches are found in the target, or reduced to zero length. Consider the following example of palatal fronting: /ʃɪp/ 'ship' → [sɪp]. Since the output string is not contained in the target string, the algorithm examines the left output branch, *sɪ*, which is also not contained in the target, and the right branch, *ɪp*, which is aligned with the second segment of the target string. A second derivation of the left output branch, *s*, is also not found in the target, and the mapping process ends with all target segments, except for *ʃ*, being aligned with output segments, and all output segments, except for *s*, being aligned with target segments.

At the end of the mapping procedure, the algorithm marks the aligned target segments as being faithfully produced and goes on to analyze the remaining segments.

It is important to note that this procedure can occasionally bring to misidentification. Consider the following case of metathesis in child's Dutch: /kɪp/ 'chicken' → [pɪk] ([2]). Although the mapping algorithm detects the output consonants *k* and *p* in the target, it would be misleading to mark them as being faithfully produced. To detect such problematic cases, the algorithm compares the mapping results of the left and right output branches. In this case, the right output branch *k* is aligned with the first target segment, and the left output branch *p* is aligned with the third target segment. In other words, the branches are transposed, and the algorithm will not mark their segments as being faithfully produced.

The mapping procedure described here is somewhat different from other common approaches to phonetic alignment, such as the dynamic algorithm for calculating minimal edit distance (see [5] for a review of other algorithms). The current proposal starts by comparing maximal strings and works its way down only if necessary. This approach is more economical than calculating the minimal edit distance table, since it does not consider a-priori irrelevant solutions. Note, however, that this advantage holds as long as the target and output resemble each other to a reasonable degree. In addition, the current approach dissociates the characterization of phonological changes from the identification of global similarity. By contrast, edit distance calculations work on segment-size units, and are thus concerned with the nature of phonological processes from the beginning.

Finally, the current proposal works on unsyllabified tokens. This is because the presence of prosodic markers may complicate the mapping (e.g., when the syllabic structure is altered due to deletions/insertions). Therefore, at the beginning of the procedure, all prosodic markers are removed, and pasted back after the identification process is completed. Other systems perform the detection on syllabically-aligned target-output forms (see for example, [3] and [4]).

2.2. Minimized weighted differences

After mapping is completed, the algorithm examines the non-aligned segments in attempt to find the best description for the target-output differences. First, the sets of features of all non-aligned segment are obtained from the system's phonetic table. Then, each non-aligned target segment is paired with each non-aligned output segment and their relation is examined in terms of differing features. In addition, each target segment can be paired with an *indel* (null segment) to simulate a case of deletion. Similarly, each output segment can be paired with an indel to simulate epenthesis.

To estimate the likelihood of each possible target-output segment alternation, the algorithm calculates the *feature distance* between the segments according to the number of features that separate the segments.

A second likelihood quantity, *position distance*, is calculated to reduce the chances of false pairing, e.g., mistaking an assimilated segment for its trigger (for example, aligning the first output *g* in /dɔg/ 'dog' → [gɔg] with the target *g* rather than with the target *d*). To determine the position distance, we first calculate the *anchor*, which is the extrapolated position of the first output segment in the target. Consider the Hebrew target-output pair /tapuaχ/ 'apple' → [buaχ]. The mapping procedure aligns the output sub-string 'uaχ' to position 4 in the target. The first aligned output segment, *u*, is the second output segment, and therefore the extrapolated position of the first (unmapped) output segment in the target is 3 (indeed, the first output segment *b* results from the voicing of the third target segment *p*). The position distance between unmapped output and target segments is calculated with reference to the anchor. In the example above, the position distance between the first output segment *b* and the first target segment, *t*, is 2. Therefore, the output *b* is more likely to be paired with the target *p* (position distance 0) than with the target *t*. The position distance between a segment and an indel depends on the position of the indel. In the above example, two indels are added at the beginning of the output ([##buaχ]). Thus, the target *p* is at a distance of 1 from the nearest indel, which makes deletion a less likely description of the fate of *p*, compared to voicing (alignment with output *b*).

After calculating the position and feature distances between all target-output segment pairs, the algorithm finds the minimal weighted differences over all legal pair combinations to determine the most likely full descriptions of target-output relations (legal combinations are those that match all target segments, such that two target segments cannot be paired simultaneously with the same output segment).

The optimal description of target-output relations may include some context-free substitutions. After finding all the features that separate a target segment S_t from an output segment S_o, the algorithm searches the table of phonological processes of the analyzer for a process that matches the situation. Substitutions are defined in the table in terms of the output value of a relevant feature, conditions that constrain the target and the output segments, and additional obligatory/optional changes. For example, identification of *gliding* (e.g., /lʊk/ 'look' → [wʊk]) requires pairing a target liquid ([+LIQ]) with an output glide ([C,-CONS]). Thus, the phonological processes table contains the value '-CONS' in the 'Result' column, and the value '+LIQ' in the 'Condition on

target' column. However, in addition to a difference in the value of the feature [CONS] the two segments may also differ in other respects, such as place and manner of articulation. These differences must be indicated in the table to allow correct identification.

After identifying context-free substitutions, the algorithm compares the output alignments of altered target segments with other output segments to detect possible cases of assimilation (comparisons are made only with respect to altered target features).

The algorithm described here was tested on the data of the Hebrew monolingual child RM that was recorded as part of the *Child Language Project* at Tel-Aviv University ([1]). The algorithm achieved high accuracy rates in the detection of phonological processes, and is capable of analyzing tens of thousands of tokens in several minutes.

2.3. Diacritical operations

Diacritical marks are treated as operators, changing feature values of their hosting segments. When feature sets of target/output segments are collected for the identification of phonological processes, these sets are modified according to the nature of present diacritics. Each operator is defined in terms of the affected feature(s), the type of operation, and its content. There are four types of operators:

(i) Add: adds content to a given tier. For example, the ejective symbol (e.g., k') adds the glottis as a second place of articulation of the hosting segment (e.g., [VELAR-GLOTTAL]).

(ii) Change: changes the valence of a feature. For example, the aspiration sign (e.g., t^h) sets the value of the [SG] tier to [+SG].

(iii) ChangeFlex: the effect of such operators depends on the nature of the hosting segments. For example, the dental sign (e.g., t̪) changes the place of articulation of alveolars to [DENTAL], and that of bilabials to [LABIODENTAL].

(iv) Combine: these operators characterize affricates and doubly-articulated consonants. In the case of affricates, it sets the manner of articulation of the combined segment to [AFFRICATE], and keeps only the value of the release portion of the affricate in all the other tiers. For doubly-articulated consonants, the *combine* operator combines the places of articulation of both segments into a single representation. In other tiers it keeps only one value. For opposing binary values, the '+' value is kept. Thus, k͡p is represented as [BILABIAL-VELAR] in the place tier, as [+LAB] in the [LAB] tier, and as [+DOR] in the [DOR] tier.

The system described in this paper contains an editable table in which all diacritical marks are listed and defined. A second table is dedicated to *flexible operators* (i.e., defined by the ChangeFlex operation).

3. PHONOLOGICAL QUERIES

The query algorithm finds all tokens containing a desired sequence of phonological entities. The sequence can be any combination of segments and feature complexes. For example, the query string *V[-SON,-CONT]* is interpreted as any vowel followed by a non-continuant obstruent.

3.1. Basic algorithm

The query algorithm has two main stages: the first stage involves decoding of the query string and preparing the corpus for the query. The query is performed in the second stage.

3.1.1. Decoding and corpus conversion

The decoder part of the algorithm identifies the requested phonological units in the query string by searching sub-parts of the string in the phonological database. Extraction of sub-parts is performed using the function *mid(x,n,m)* which returns *m* characters of the string *x* starting from character *n*. A feature complex is detected by the occurrence of the feature opening marker (e.g., '['). If such a marker is found, the algorithm searches for the complex end marker (e.g., ']'). In addition, sub-parts of the complex are identified by occurrences of the in-feature marker (e.g., a comma) between feature complex boundaries. Then, the algorithm can extract and identify the features enclosed by the brackets with the assistance of the phonetic table.

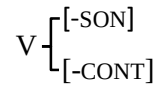
Following the decoding of the query string into phonological units, the algorithm finds the values of the desired features for each segment in the phonetic table, and then creates representations for all the tokens in the corpus in terms of those features. These representations are written to the spreadsheet alongside the original tokens. Thus, for the above example, the algorithm creates three representations of the corpus tokens in terms of CV units, and the values of the [SON] and [CONT] features. These representations are demonstrated for the token /dɔg/ 'dog' in Table 1:

Token	**CV**	**[SON]**	**[CONT]**
dɔg	CVC	[-S][+S][-S]	[-C][+C][-C]

Table 1: Representations of /dɔg/ as sequences of CV units, [SON] and [CONT] features ([S] and [C] abbreviations are used for space considerations).

The query decoder also creates a scheme according to which the query will be performed. The scheme is a hierarchical organization of the query string sub-units by their linear order. The scheme for the example above is illustrated in Figure 1. It has two levels corresponding to the two major units of the query string, and the second level has two nodes corresponding to the two features in the complex.

Figure 1: Query scheme for V[-SON,-CONT]



The meaning of this organization is that the algorithm will return the intersection of all tokens containing a sequence of a vowel and an obstruent, and the tokens containing a sequence of a vowel and a non-continuant segment

The scheme is also constrained by positional specifications: the search can be limited to word initial, word middle, or word final positions, and can also be matched with whole tokens. To facilitate the query, the algorithm uses the feature sequences (Table 1) to filter the appropriate columns and exclude irrelevant tokens.

3.1.2. Querying

The query is performed as follows: the algorithm examines all the sequences in the first filtered column (e.g., the CV column in Table 1) and extracts tokens containing the first query section (e.g., *V*) in the desired position.

Then, the algorithm examines the equivalent sequences in the subsequent column(s) searching for the second query section. In the above example, it examines cells in the third and the fourth columns of Table 1 for occurrences of [-SON] and [-CONT].

Searching from the second section onwards is constrained by the requirement that each section is continuous with the preceding one. To ensure continuity, the algorithm calculates the lengths of the examined strings in terms of characters as well as phonological units. In columns of feature decompositions, the length in phonological units up to a certain position is the number of feature opening markers (e.g., '[').

Consider, for example, a word-final query of *V[-SON,-CONT]* and the token /dɔg/. First, *V* is found in position 2 of the sequence *CVC* in column 2 of Table 1. Then, the algorithm calculates a length of 2 CV units from the beginning of the sequence to the position in question. Next, both sub-parts of section 2 (e.g., [-SON], and [-CONT]) are searched for in columns 3 and 4. According to the results of the previous iteration, section 2 is required to begin at the third phonological unit whose position equals the position of the third '[' marker. If the desired sub-parts of section 2 are found in the appropriate positions in the relevant tiers (which is the case for the example in question), the algorithm will continue to the next sections (if exist).

Finally, when reaching the last query section, the algorithm checks its end position to determine whether it satisfies the positional constraint of the query (e.g., in mid-word queries the last section must not be aligned with the end of the token). If all query sections are found and satisfy the positional constraint, the examined token will be recorded and counted.

The query algorithm can detect multiple occurrences of the query string within a single token. For example, /ˈlɪli/ 'lily' contains two occurrences of the sequence [+LAT][V,+HIGH]. When the first section (e.g., [+LAT]) is found in some cell of the corpus, the algorithm searches for other occurrences of that section within the same string. Then, the querying procedure described above is performed separately for each of these instances. Separate counters are maintained for the number of tokens containing the query string, and the total occurrences of the string in question. In the current example, if no positional constraints are placed, then /ˈlɪli/ will contribute one occurrence for the token counter, and two occurrences for the string counter.

3.2. Diacritical operations

Diacritical marks also affect the querying process. Before performing the query, the algorithm cleans all feature tiers from the diacritical marks (which were present in the original token list, and were left untouched by the conversions of segments to feature representations). However, in tiers relevant for diacritical operations, the algorithm modifies the representation of the hosting segments, according to the nature of the operators.

After the corpus is converted to the appropriate feature representations, the algorithm searches each tier for the presence of relevant diacritical marks. If such marks are found, the diacritical operations are executed. Thus, for /ˈpi͡tsa/ 'pizza' (Hebrew), the place and manner tiers after diacritical operations will be as in Table 2:

Place	**Manner**
[BIL][V][ALV][V]	[PLO][V][AFF][V]

Table 2: Representations of /ˈpi͡tsa/ as sequences of place and manner features (BIL = bilabial; ALV = alveolar; PLO = plosive; AFF = affricate; V = vowel).

4. ACKNOWLEDGEMENTS

I would like to thank the audience of the Israeli Phonology Circle (2012, 2014) for their comments and suggestions. I would especially like to thank the following people: Outi Bat-El, Evan Cohen, Galit Adam, Stav Klein, and Hadas Yeverechyahu. Finally, I would like to thank Roni Gafni for inspiring this project.

5. REFERENCES

[1] Bat-El, O. 2014. *The Acquisition of Hebrew Phonology and Morphology*. Brill.

[2] Fikkert, P., Levelt, C. 2008. How Does Place Fall Into Place? The Lexicon and Emergent Constraints in the Developing Phonological Grammar. In: Avery, P., Dresher, B.E., Rice, K. (eds.), *Contrast in phonology: Perception and Acquisition*. Berlin: Mouton, 231-268.

[3] Gedge, J., Hedlund, G., Rose, Y., Wareham, T. 2007. Natural Language Process Detection: From Conception to Implementation. Paper presented at the 17th Annual Newfoundland Electrical and Computer Engineering Conference (NECEC), St. John's NL.

[4] Hedlund, G.J., Maddocks, K., Rose, Y., Wareham, T. 2005. Natural language syllable alignment: From conception to implementation. In: *Proceedings of the Fifteenth Annual Newfoundland Electrical and Computer Engineering Conference.*

[5] Kondrak, G. 2003. Phonetic alignment and similarity. *Computers and the Humanities*, 37(3), 273-291.

[6] Rose, Y., MacWhinney, B., Byrne, R., Hedlund, G., Maddocks, K., O'Brien, P., Wareham, T. 2006. Introducing Phon: A Software Solution for the Study of Phonological Acquisition. In: Bamman, D., Magnitskaia, T., Zaller, C. (eds.), *Proceedings of the 30th Annual Boston University Conference on Language Development*. Somerville, MA: Cascadilla Press, 489-500.

[7] Rose, Y., MacWhinney, B. 2014. The PhonBank Project: Data and Software-Assisted Methods for the Study of Phonology and Phonological Development. In: Durand, J., Gut, U., Kristoffersen, G. (eds.), *The Oxford Handbook of Corpus Phonology*. Oxford: Oxford University Press, 308-401.