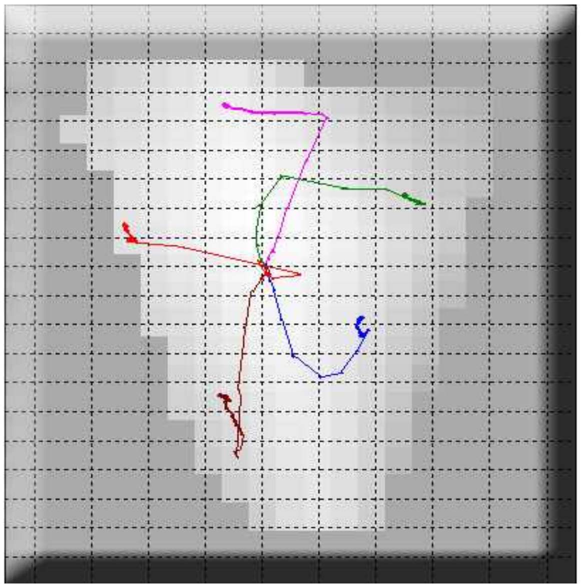
Emergent optimal vowel systems



Jan-Willem van Leussen, 0240184
MA thesis for the Master's in General Linguistics
University of Amsterdam
December 2008

Supervisor: prof. dr. Paul Boersma

Second reader: dr. Bart de Boer

Table of contents

Figure 1 was adapted from Schwartz et al (1997b).

Figures 8ab were adapted from de Boer (2001).

Figure 26 was taken (with permission) from Boersma (2008).

Figures 9, 11, 15, 16 and 18-25 were created using Praat (Boersma & Weenink 2008).

Abstract

An interesting aspect of vowel systems is that they seem to balance between articulatory ease and auditory contrast. This tension is often proposed as the cause of the remarkable overlap between the organization of vowels in various languages. This thesis aims to integrate self-organizational, agent-based models of vowel dispersion with an existing Optimality Theoretic model of non-teleological phoneme dispersion. To this end, a computer simulation combining both approaches was developed. Simulation results show that dispersed vowel systems still emerge in the artificial language of the agents, although its predictions with respect to vowel quality are not entirely accurate. The ability of the model to account for the diachronic process of chain shifts caused by vowel splits or mergers is also explored. The results confirm that innate constraints are not needed to model vowel dispersion and that these types of simulations may be helpful in investigating synchronic and diachronic phonological phenomena. However, the model described in this thesis needs to be enriched with more levels of representation to increase its explanatory power.

1 Introduction

Anyone who has ever studied a second language is aware of the fact that different languages organize things differently. This holds true for their sound systems as well: a phoneme in one language may be pronounced differently in another, or is perhaps not considered a phoneme at all. On the other hand, it has long been remarked by people who have studied a *lot* of languages that the variation between languages is not completely arbitrary: of all the sounds that can be used as phonemes, some are found in almost all languages while others are extremely rare crosslinguistically. These similarities in the organization of sound systems even occur between languages that have not been proven to be related.

Apparently languages, or rather the people that speak them, prefer certain sounds to others. But the origin of these preferences has long been a matter of controversy, and not just in the area of phonology. Adherents to the Chomskyan theory of Universal Grammar propose that these preferences are to a certain extent innate. On the other hand, *phonetically-based* or *substance-based* explanations tend to attribute the regularities of sound systems to properties of the sounds and the means through which they are communicated. Over the last few decades, this last type of hypothesis has gained ground through the new methodology of simulation by computer. More recently, formal phonological theory has also been adapted to explain the variation and similarity found in sound systems.

This thesis describes a computational model that aims to explain some synchronic and diachronic properties of the patterns found in vowel systems around the world. The model that is used combines approaches from two disciplines: the methodology of *agent-based* modelling from the field of Artificial Intelligence, and the linguistic framework of Optimality Theory. I hope to show that such an interdisciplinary approach is fruitful and plays to the strength of both approaches.

The thesis is structured as follows: in Chapter 2 I will give an overview of the tendencies found in vowel inventories. Chapter 3 serves as a quick overview of Optimality Theory, which is central to the description of the simulation in Chapter 4. The outcome of the simulations and the effects of varying its parameters are described in Chapter 5; in Chapter 6 these results are interpreted and discussed, along with suggestions for adaptations and improvements of the model.

2 Vowel systems

2.1 Tendencies in vowel systems

The phonological inventories of spoken languages show remarkable variety, both in terms of the number of phonemes that is used and in terms of which phonemes are used. Pirahã, a language spoken by a few hundred people in the Brazilian Amazon basin, is claimed to be the language with the smallest phoneme inventory, namely three vowels and seven consonants for female speakers (Everett 2005). On the other end of the spectrum is the Khoisan language Xóõ, spoken mostly in Botswana, which according to Traill (1985) distinguishes well over a hundred phonemes, among which more than 80 click consonants.

The phoneme inventories of the majority of languages fall somewhere between these two extreme examples. In fact, perhaps just as striking as the variety found in phonological systems is the regularity that can be found in them. Figure 1, adapted from Schwartz et al. (1997b), gives an overview of the most common 3 to 7 vowel systems containing from 3 to 7 vowels found in the UPSID (UCLA Phonological Segment Inventory, Maddieson 1984).

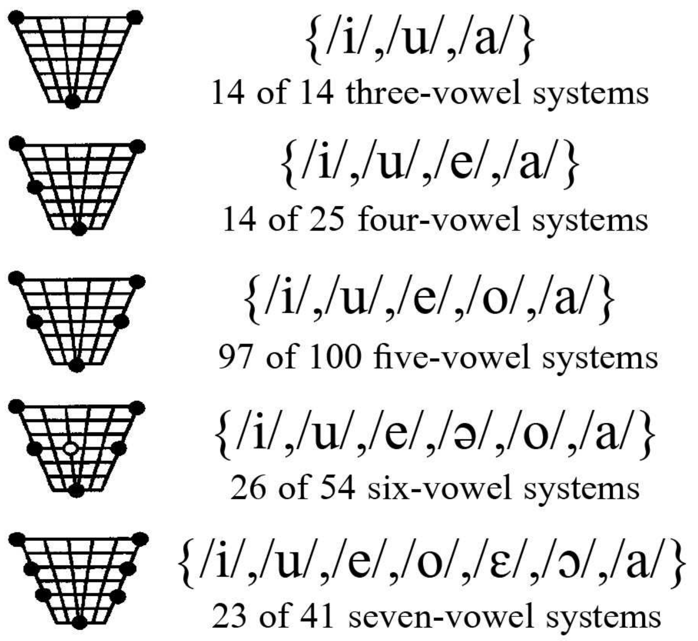

{/i/,/u/,/a/}
14 of 14 three-vowel systems
{/i/,/u/,/e/,/a/}
14 of 25 four-vowel systems
{/i/,/u/,/e/,/o/,/a/}
97 of 100 five-vowel systems
{/i/,/u/,/e/,/ə/,/o/,/a/}
26 of 54 six-vowel systems
{/i/,/u/,/e/,/o/,/ɛ/,/ɔ/,/a/}
23 of 41 seven-vowel systems

Figure 1, adapted from Schwartz et al (1997b). Most frequent n-vowel systems found in the UPSID database and their frequencies, for n = 3 to 7.

The figure shows that languages display a preference for certain vowels and configurations within the available vowel space. All three-vowel languages in the database are made up of the three corner vowels /a/, /i/ and /u/; and almost all larger inventories include at least these three vowels. Inventories with five vowels are the most popular, and of these an overwhelming majority is of the type /a/, /e/, /i/, /o/, /u/. More generally, there seems to be a preference for vowels located at the periphery of the vowel space; and there is a clear tendency for symmetry in the vowel space, which is most pronounced in the systems containing an odd number of vowels.

In linguistics, these phonological universals are often described in terms of markedness: certain acoustic or articulatory features of sounds are more marked than others. Marked sounds occur more rarely, and usually the existence of a marked phonological feature in a language implies the existence of its unmarked counterpart. For example, the close front unrounded vowel /i/ is less marked than its rounded counterpart /y/; as a rule, languages that have /y/ also have /i/, but not all languages with /y/ have /i/. Implicational rules often describe the tendencies that are seen in phonological inventories, but fail to explain *why* certain sounds are more marked than others. According to the Chomskyan theory of Universal Grammar, some aspects of markedness derive from innate properties of the mental language ability (see for instance Chomsky & Halle 1968).

Another type of explanation seeks to derive common phonological systems from the phonetic properties of these phonemes. As can be seen in Figure 1 above, vowels tend to be at the periphery of the acoustic vowel space that people are able to produce. From this, it may be deduced that there is a tendency in languages to position phonemes as far away from each other as possible in the available acoustic space. There is a clear advantage to this: it reduces the possibility of one phoneme being erroneously perceived as another, and thus increases the communicative efficiency of a language. On the other hand, there also seems to be a tendency in the opposite direction: languages with only a single phoneme along a given acoustic continuum usually have this phoneme in the center of this continuum, and when there are more phonemes along this continuum they will be placed increasingly farther towards the periphery (Boersma & Hamann 2008). The Quantal Theory of Stevens (1972) also approaches tendencies in vowel systems from an articulatory perspective. Stevens observes that the vowel phonemes /a/, /i/ and /u/ are more robust to small deviations in articulation than others, and introduces the term *quantal* for this property. Generally, it appears that vowel systems (and phoneme inventories in general) can be characterized as balancing between minimal articulatory effort and minimal acoustic confusion (e.g. Ten Bosch 1991).

Because these types of explanations describe relationships between phonemes in terms of their quantitative phonetic properties, calculating what vowel systems are considered optimal under these circumstances quickly becomes very complex. The increase in computing power that the last decades have seen has allowed researchers to employ computer models for the investigation of phonological systems. The next section presents an overview of what has been achieved with these models.

2.2 Simulating optimal vowel inventories

In a pioneering study, Liljencrants and Lindblom (1972) built a numerical model that, for n vowels in the acoustic vowel space, tries to maximize the sum of the perceptual distances between these vowels. The results closely resemble common vowel systems, especially for lower values of n. Since its publication, some refinements have been made to the basic model of Liljencrants and Lindblom, allowing for more accurate predictions. For instance, Schwartz et al. (1997b) introduced besides the tendency for dispersion, also a tendency toward focalization, a measure of spectral saliency intrinsic to vowels, similar to the concept of quantality of Stevens (1972). By systematically varying the weight of these two parameters in the model, they were able to find the values for the dispersion and focalization parameters that best predicted phonological data.

Nevertheless, these purely substance-based models still fall short of explaining *how* this optimization takes form. After all, speakers of a language do not consciously strive toward changing its phonological system. Rather, optimization seems to arise as a side effect from the mechanisms involved in communication and language learning. De Boer (2001) proposes *self-organization* as a key concept in explaining common sound patterns. Self-organization is a phenomenon ubiquitous in nature, where order on a large scale emerges from interactions on a small scale. De Boer shows that in computer simulations based on a population of agents interacting by playing a vowel 'imitation game' and articulating through a model of the human vocal tract, realistic vowel systems emerge even though the agents themselves do not have vowel dispersion as an explicit goal; they are a consequence of the agents' means of communication and the noisy circumstances under which this communication takes place. He argues that this eliminates the need for proposing innate preferences for certain types of vowels above another.

Attempts have also been made to integrate these phonetic accounts of phonological dispersion into existing linguistic and phonological theory. Flemming (2002) and Steriade (2001) both give Optimality Theoretic accounts using constraints that express contrast or auditory distance between phonemes. Boersma & Hamann (2008) show that even without explicit constraints on phonological contrast, dispersion effects can be shown to result from learning if *bidirectionality* of constraints is assumed. By using a mechanism where the OT output of a learner, under the influence of some noise, forms the basis for the input of another learner, they demonstrate the emergence of dispersion and related effects like chain shifts in sibilant inventories. Wedel (2006) has shown that phonological and other types of contrasts emerge and are maintained in exemplar-based theories of language acquisition.

2.3 Proposed model

The model proposed in this thesis is an attempt to integrate self-organizing agent-based models such as De Boer (2001) into the linguistic framework of Optimality Theory. Specifically, the perception and production of vowels of agents in the simulations is governed by bidirectional stochastic OT grammars, as in Boersma & Hamann (2008) and also for vowels in Boersma (2007). Rather than representing a generation of speakers by a single OT grammar, speakers and listeners will be represented as independently acting agents that perceive, learn and speak through their own ranked OT grammars. Such an agent-based approach to OT is arguably a more realistic approach to interaction and language acquisition. Furthermore, I will argue that modeling populations of speakers can be used to pursue hypotheses of language variation and change in a way that is not possible in Boersma & Hamann's original model.

Before turning to a description of the model proper, the next chapter will explain the basics of Optimality Theory needed to understand the workings of the model.

3 Optimality Theory

This chapter will provide a rather brief introduction to the paradigm of Optimality Theory (OT), and to Stochastic OT and the Gradual Learning Algorithm, with the purpose of giving an overview and background of the model for vowel perception, production and learning used in the simulations described in this thesis. Readers familiar with Optimality Theory and its extension in the form of Stochastic OT can safely skip to chapter 4; readers unfamiliar with OT but wishing to learn more about it may want to consult a more comprehensive work such as Kager (1999).

3.1 OT basics

Optimality Theory is a linguistic framework developed by Alan Prince and Paul Smolensky, first set out in Prince & Smolensky (1993), that has gained a lot of popularity in linguistics since its inception. While OT has been applied to various areas of linguistics (see Archangeli 1997 for an overview), the field of phonology for which it was originally developed remains the most popular application. The core idea of OT is that language use can be represented by a set of hierarchically ranked constraints that generates from an 'underlying' input form a set of outputs, and chooses an 'optimal' surface output based on the ordering of these constraints. It is presumed that the constraints themselves are universal to all languages, and can represent all possible languages; the *ranking* of the constraints is what determines the typological parameters of a specific language.

The evaluation process is usually represented with tableaus like Figure 2 below, which gives the familiar example of final obstruent devoicing. The word *bed* "bed" in Dutch has a surface structure of /.bɛt./, with a voiceless coda. However the underlying form of the word, which is also reflected in its orthography, is assumed to be |bɛd|, as can be seen in when the plural suffix +|ən| is added: *bedden* "beds" becomes /.bɛ.dən./. Figure 2 describes devoicing of final /d/ in Dutch with three constraints operating on the candidates. The first constraint MAX militates against output forms which contain fewer segments than the input form, forbidding deletion. The second constraint *VOICEDCODA forbids output candidates containing a voiced consonant in the coda. The third constraint IDENT militates against surface outputs which are not identical to the underling input candidate. Prince and Smolensky distinguish between two basic types of constraints: *markedness constraints*, which concern only the form of the output, and *faithfulness constraints* which concern the relation between input and output forms. In this example, MAX and IDENT are faithfulness constraints, and *VOICEDCODA is a markedness constraint.

|bɛd|	MAX	*VOICEDCODA	IDENT
/.bɛd./		*!	
☞ /.bɛt./			*
/.bɛ./	*!		

Figure 2: an OT tableau for final obstruent devoicing in the Dutch word "bed". Asterisks indicate a violation; exclamation marks indicate a fatal violation that eliminates a candidate. The constraints are ordered by importance from left to right. A pointing finger indicates the optimal candidate. To keep things simple only three output candidates are shown, and only three constraints acting on them are modelled.

The constraints in this example are ordered MAX > > *VOICEDCODA > > IDENT, and the candidates are evaluated step-by-step in this order. When a candidate violates a constraint while one or more other candidates do not, the violation is *fatal* and the candidate is no longer taken into consideration at the next step. This process continues until a single winning candidate remains, in this case /.bɛt./.

The ordering of the constraints is crucial to the output of the grammar here. If the ordering had been MAX > > IDENT > > *VOICEDCODA, the winning candidate would have been /bɛd/. Such an ordering would describe a language like English, where there is no devoicing of final obstruents and the word "bed" is indeed structured at the surface as /.bɛd./. The MAX constraint may seem unnecessary here, but ranking it lower could for instance model the language of an infant learning to speak Dutch or English and not able to articulate codas; or the adoption of loanwords in a language that allows only /CV/ syllables.

However, different constraint rankings need not always result in a different output. Figure 3 is an OT tableau for production of Dutch *pet* "cap". Again the candidate ending in a voiceless consonant wins; but a tableau for an English grammar producing the word *pet,* with the order of *VOICEDCODA and IDENT switched, would result in the same output.

|pɛt|	MAX	*VOICEDCODA	IDENT
/.pɛd./		*!	
☞ /.pɛt./			
/.pɛ./	*!		

Figure 3: an OT tableau for the Dutch word pet *with the same grammar that was used in Figure 2. Note that in this case, the winning candidate does not violate any of the constraints and would therefore have won regardless of the constraint ordering.*

While the above tableaus describe a somewhat trivial problem, the OT approach of viewing phonological rules as competing and conflicting constraints has proven useful for describing many problems in phonology. On the explanatory side however, the framework as described above is lacking. One problem is that it is not possible to express the variation that occurs within languages and even within single speakers using fixed constraint rankings. Another problem is that tableaus with fixed rankings cannot express how language is learned, i.e. how children obtain the constraint ranking appropriate for their language from all possible rankings, depending on their language input. Several proposals for expressing learnability and variation in OT exist to alleviate these problems. One such proposal is Stochastic Optimality Theory (Boersma 1997, Boersma and Hayes 2001). The next section will describe this formalism in more detail.

3.2 Stochastic Optimality Theory

In Stochastic OT, the strict ordering of constraints is replaced by an ordering based on a real-valued *ranking value* for every constraint. At evaluation time, this ranking value is then subject to some random noise drawn from a normal distribution with a mean of 0 and a standard deviation defined in an *evaluation noise* parameter. When these stochastic evaluations take place, the constraints are ordered by their *disharmony* value, which is the ranking value plus the random noise. As a result, the ordering of constraints by their disharmony may differ from the ordering predicted by their ranking value, and the optimal output a grammar will produce for a given input is subject to variation.

Figures 4-5 adapt an example of nasal place assimilation from Boersma (1997), although the names of the constraints have been changed. IDENT, as above, is a faithfulness constraint that militates against outputs that differ from the underlying input form. ASSIMILATENASAL is a constraint that forces regressive place assimilation of a nasal to a stop following it. In this example, it changes an underlying |np| to a surface /mp/. In Figure 4, the two constraints have a true ranking of 52 and 49 respectively. This means that ASSIMILATENASAL is ranked higher than IDENT, and thus without stochastic evaluation the nasally assimilated output /.am.pa./ will be more optimal than the faithful output /.an.pa./.

|an + pa|	ASSIMILATENASAL	IDENT
	52.0	49.0
/.an.pa./	*!	
☞ /.am.pa./		*

Figure 4: An OT approach to nasal place assimilation. Constraint ranking values are written below them. Ordered by their true ranking value, the constraints select /.am.pa./ as the optimal candidate.

Now consider figure 5, in which evaluation noise has been added, drawn from a normal distribution with a standard deviation of 2.0. Under these evaluation noise settings, the disharmony (ranking value plus noise) of a constraint with a true ranking of 52 will lie between 50 (one standard deviation below the mean) and 54 (one standard deviation above the mean) approximately 68% of the time. In this tableau, the resulting disharmonies are such that the two constraints have switched places in the ranking, allowing the candidate /.an.pa./ to win.

|an + pa|	IDENT	ASSIMILATENASAL
	49.0	52.0
	51.4	*50.7*
☞ /.an.pa./		*
/.am.pa./	*!	

Figure 5: place assimilation in stochastic OT. The disharmony of a value is shown in italics below the true ranking value. Here, the evaluation noise has placed IDENT higher than ASSIMILATENASAL.

The distribution of different possible outputs for a stochastic grammar is determined by the ranking values and the evaluation noise variable. This example, with evaluation noise of 2.0 and rankings of 52 and 49 for ASSIMILATENASAL and IDENT respectively, will result in an output distribution where nasal place assimilation is more likely than non-assimilation: /.am.pa./ wins 86% of the time and /.an.pa./ wins 14% of the time (see Boersma 1997 for how these distributions may be calculated). If both ranking values are exactly equal, the expected output distribution will be 50/50. In this way, Stochastic OT is able to model variation of outputs within a grammar, which is impossible in original OT with strict constraint rankings. Boersma (1997) argues that stochastic evaluation is necessary to explain not only variation in language, but also to explain robust language acquisition in the face of such variation. He proposes a Gradual Learning Algorithm (GLA) that can model this acquisition of a stochastic grammar.

3.3 The Gradual Learning Algorithm

Sticking with the nasal assimilation example of the previous section, a situation can be imagined where an infant is learning the language in which underlying |anpa| is assimilated to /ampa/ 86% of the time. In keeping with the OT idea of language acquisition as finding a language-specific ordering for the universal set of constraints that people are born with, the infant may start out with a different ranking in its grammar than the adult language speakers in its environment, for instance one where IDENT is ranked higher than ASSIMILATENASAL. Learning then consists of changing the constraint rankings, i.e. updating the ranking values of these constraints, until they reflect the language environment of the learner.

The GLA handles this learning through incremental, error-driven updating of the ranking values. At evaluation time, GLA-driven learners have knowledge of the candidate that *should* have won, and compare it to the output of their own stochastic grammar. Whenever the winning output form differs from this intended output form, the constraint rankings responsible for this error are updated by a small value. Figure 6 gives a tableau of this learning step occurring for the aforementioned non-assimilating infant. The infant, upon hearing a production of the underlying form |an+pa| being realized as /ampa/, checks this against its own stochastic grammar and finds that the winning candidate there is different. In response, it *demotes* the constraint(s) that were fatally violated by the candidate that should have won, and *promotes* the constraint(s) that the winning candidate violated. The size of the increment by which the ranking values are raised and lowered is defined by the *plasticity* parameter, which is 0.01 in this example.

|an+pa| /.am.pa./	IDENT	ASSIMILATENASAL
	50 → **49.99**	42 → **42.01**
	50.4	*43.1*
☞ /.an.pa./		← *
✓ /.am.pa./	*! →	

Figure 6: A learning step in the Gradual Learning Algorithm. The check mark represents the candidate that should have won in the language the learner is acquiring. The constraints responsible for the error are shifted to slightly decrease the likelihood of this error. Arrows show the direction of this shift in the constraint ranking.

While this single learning step has little effect on the output distribution of the grammar, Boersma (1997) shows that over a large number of iterations, the learner's grammar stabilizes at a point where it produces the same output that was fed to it – in this case, the rankings will then probably converge to a point where the difference between the ranking values of IDENT and ASSIMILATENASAL is 3, and the simulated infant will have learned the language of its environment.

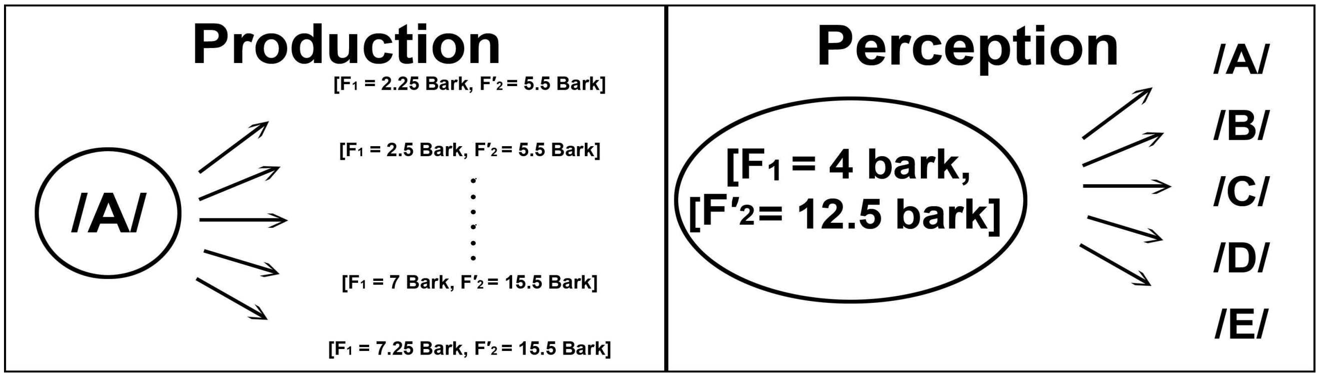
Stochastic OT combined with the Gradual Learning Algorithm has not only been applied to phonological production, but has also been used to model different aspects of speech perception and production. Some examples are cue learning in L2 acquisition (Escudero & Boersma 2003), the learning of phonological categories from phonetic input data (Boersma, Escudero & Hayes 2003) and deriving underlying stress patterns from surface forms (Apoussidou 2006). Most relevant to the current research, Stochastic OT/GLA has been used to show that certain phonological and universal rankings can be explained as *emergent* (Boersma & Hamann 2008, Boersma 2008); these rankings result from the nature of the input data and noise in the transmission of said data. These is no need for resorting to an innate preference for 'unmarked' constraint rankings or to teleological constraints to arrive at a universal typology.

4 Description of the simulation

This chapter presents the underlying framework for the computer simulations of vowel inventories discussed in this thesis. There are two crucial aspects to this framework. First, it is *agent-based*, meaning that language is represented and shaped by a number of independently acting simulated speakers communicating with one another. This agent-based approach to predicting vowel systems is largely inspired by De Boer (2001).

Secondly, the agents communicate with each other through a *bidirectional* Optimality Theoretic grammar, which is to say that perception and production of vowel sounds are determined by the same ranked constraint grammar. Boersma & Hamann (2008) have used bidirectional OT to predict the structures of sibilant inventories, and the architecture of their model has been adapted where possible to the domain of the simulation described herein.

This simulation is only concerned with vowel phonemes and their quality in terms of formant frequencies. Therefore, the language that the agents in the simulation speak is a rather abstract and impoverished one: the agents communicate simply by uttering vowel phonemes to one another. The quality of these vowels is described as a pair of values [F_1, F'_2], which stand for the first formant and *effective second formant* (Bladon 1983) respectively. Production is a mapping process that takes as input a vowel phoneme from the set of possible phonemes, and outputs an [F_1, F'_2] value pair representing the auditory realization of the phoneme. Perception is simply the inverse of the production function: it maps a perceived sound to one of the phonemes present in the grammar (Figure 7)


Production
/A/
[F1 = 2.25 Bark, F'2 = 5.5 Bark]
[F1 = 2.5 Bark, F'2 = 5.5 Bark]
[F1 = 7 Bark, F'2 = 15.5 Bark]
[F1 = 7.25 Bark, F'2 = 15.5 Bark]
Perception
[F1 = 4 bark, [F'2= 12.5 bark]
/A/
/B/
/C/
/D/
/E/
Figure 7: Schematic example of production and perception of vowels in the system. Both processes have a single input and a set of possible outputs. The actual winning output is determined by the stochastic OT grammar.

The outcome of both mapping processes is determined by a single stochastic Optimality Theoretic grammar. More precisely put, all possible outputs for a given input are generated as candidates that are evaluated by this grammar, and the ranking of the constraints in the grammar determines the most likely winning candidate. The next section will describe the agents' grammars in more detail.

4.1 Constructing the grammars

Parameters of the grammar

As said, the simulation represents the quality of vowel phonemes in terms of two numerical variables: the frequency of its first formant F_1, and the frequency of its effective second formant F'_2. As in Boersma & Escudero (2008), these originally continuous formant values are discretized. The F_1 may take on a value from 2.25 to 7.25 Bark, discretized in 0.25 Bark steps; the F'_2 takes a value from 5.5 to 15.5 Bark, discretized in 0.5 Bark steps. Since both F_1 and F'_2 have 21 distinct possible values in the grammar, this means that there are ($21^2 =$) 441 different [F_1,F'_2] value pairs that can be associated with a phoneme.

The number of phonemes available to the agents is given at the start of the simulation, and the phonemes are simply named after consecutive letters of the alphabet: for instance if the language is set to have three phonemes, they will simply be named /A/, /B/ and /C/. Please note that these vowel labels are arbitrary and do not in any way predispose the agents toward pronouncing the vowels one way or another; it is precisely the point of the simulations described here that such predilections are not needed. Also note that the number of phonemes available does not change during the simulation. Investigating phonological mergers and splits would be interesting, and it is desirable that a full-fledged model of diachronic phonological change and variation incorporates and predicts these phenomena; however the processes behind vowel mergers and splits are most likely too complex to simulate in the impoverished vowel-only languages of this simulation. On the other hand, it *is* possible to simulate a language in which a merger or split has just taken place, and this will in fact be done in section 5.4.

Constraints and candidates

Using the parameters described in the previous paragraph, the constraints and candidates that make up the OT grammar and its outcomes can be defined. A candidate in this grammar is simply a phoneme from the set of available phonemes linked with a [F_1, F'_2] value pair in Bark. An example candidate is /A/, [$F_1 = 4$, $F'_2 = 10$], representing an instance of the phoneme /A/ with the quality determined by these particular formant frequencies.

The candidates are processed by the grammar by checking them against negatively formulated constraints, which come in two types (as in Boersma & Hamann 2008). First, there are *cue* constraints (Escudero & Boersma 2004), which militate against a certain F_1 *or* F'_2 value being perceived and produced as a certain phoneme, e.g. "a candidate with an F_1 of 4 Bark is NOT phoneme /A/", represented as *[$F_1 = 4$] /A/. Then there are *articulatory* constraints (Kirchner 1998, Boersma 1998), which militate against certain [F_1, F'_2] pairs being produced, e.g. "do not produce a candidate with an F_1 of 2.75 Bark and an F'_2 of 10 Bark", represented as *[$F_1 = 2.75$, $F'_2 = 10$]. All constraints in the grammar are associated with a real-valued *ranking value*, which determines the order in which they evaluate the candidates (see chapter 3.2 on Stochastic OT).

An important distinction must be made between the cue constraints and articulatory constraints in an agent's grammar. Cue constraints represent the learned, phoneme-specific aspects of speaking a language, which is determined by the input a person receives and therefore differs between languages and, to a lesser extent, between speakers of a language. For a newly born agent, the cue constraints are initially all ranked equally at 100.0; however, due to the learning process described in section 3.3, these rankings will change over time, reflecting the input an agent has received.

On the other hand, the articulatory constraints represent aspects of speech production which are constrained by the anatomy of the human vocal tract. While sizes and shapes of vocal tracts may vary considerably between speakers, the assumption is made here that for speaking adults, the space of vowel sounds that can be produced and the effort required to produce the vowels within this space does not vary (i.e. the problem of speaker normalization is not considered in these simulations). This invariance is reflected in the fact that the rankings of the articulatory constraints are the same for all agents. They are also static: the learning step which updates the cue constraint rankings does not apply to the articulatory constraints.

Determining the articulatory constraint rankings

In Boersma & Hamann (2008), the ranking of the articulatory constraints corresponds directly to their distance from the center of auditory spectrum, assuming that it corresponds to articulatory effort. For the two-dimensional vowel continuum of this simulation, this assumption would be too simplistic: merely taking the distances from the F_1 and F'_2 values of the neutral vowel /ə/ as indications of effort rankings would result in an unrealistic 'round' vowel space. A direct mapping from auditory values to articulatory effort will therefore not suffice here.

To be able to come up with ranking values that reflect articulatory possibility and effort, I use the simple articulatory model from De Boer (2001), for which the calculations are in turn interpolated from values that Vallée (1994) generated with the complex articulatory model of Maeda (1989). De Boer's articulatory model takes as input three parameters *p*, *h* and *r*, which take on a value between 0 and 1, standing for tongue position (back to front), tongue height, and lip rounding respectively. For the height and position parameters *h* and *p*, 0.5 represents a 'neutral' stance of this articulatory dimension; for the lip rounding dimension *r,* the value $r = 0$ represents minimal rounding, whereas $r = 1$ represents maximal rounding. Thus an input of ($p = 0.5$, $h = 0.5$, $r = 0$) results in a schwa-like vowel. Figure 8a reproduces the formulas from de Boer (2001) for mapping these inputs to values for the first four formants. The second to fourth formants are then collapsed into a single value F'_2..

(Figure 8b). Figure 9 shows the outcome for a set of vowels and their associated height, position and rounding values.

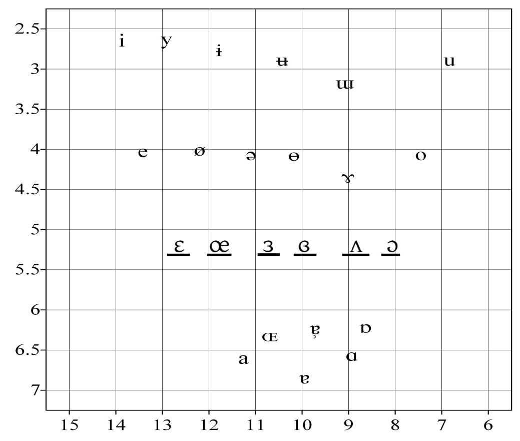
$$
\begin{aligned}
F_1 = {} & ((-392+392r)h^2 + (596-668r)h + (-146+166r))p^2 \\
& + ((348-348r)h^2 + (-494+606r)h + (141-175r))p \\
& + ((340-72r)h^2 + (-796+108r)h + (708-38r)) \\[1em]
F_2 = {} & ((-1200+1208r)h^2 + (1320-1328r)h + (118-158r))p^2 \\
& + ((1864-1488r)h^2 + (-2644+1510r)h + (-561+221r))p \\
& + ((-670+490r)h^2 + (1355-697r)h + (1517-117r)) \\[1em]
F_3 = {} & ((604-604r)h^2 + (1038-1178r)h + (246+566r))p^2 \\
& + ((-1150+1262r)h^2 + (-1443+1313r)h + (-317-483r))p \\
& + ((1130-836r)h^2 + (-315+44r)h + (2427-127r)) \\[1em]
F_4 = {} & ((-1120+16r)h^2 + (1696-180r)h + (500+522r))p^2 \\
& + ((-140+240r)h^2 + (-578+214r)h + (-692-419r))p \\
& + ((1480-602r)h^2 + (-1220+289r)h + (3678-178r))
\end{aligned}
$$

Figure 8a: Calculating values of the first four formants (in Hertz) on basis of three parameters p*,* h*, and* r*. Adapted from De Boer (2001).*

$$
\begin{aligned}
w_1 &= (3.5-(F_3B-F_2B))/3.5 \\
w_2 &= ((F_4B-F_3B)-(F_3B-F_2B))/(F_4B-F_2B)
\end{aligned}
$$

$$
\begin{array}{ll}
F'_2B = F_2B & \text{if } (F_3B - F_2B) > 3.5 \\
F'_2B = ((2-w_1)F_2B+w_1F_3B) / 2 & \text{if } (F_3B - F_2B \leq 3.5) \text{ and } (F_4B - F_2B > 3.5) \\
F'_2B = ((\,(w_2F_2) + (2 - w_2)F_3B) / 2) - 1 & \text{if } (F_4B - F_2B \leq 3.5) \text{ and } (F_3B - F_2B < F_4B - F_3B) \\
F'_2B = (((2+w_2)F_3B - (w_2F_4B)) / 2) - 1 & \text{if } (F_4B - F_2B \leq 3.5) \text{ and } (F_3B - F_2B \geq F_4B - F_3B)
\end{array}
$$

Figure 8b: Calculating the effective second formant F'_2 *from the second, third and fourth formants. The Bs indicate that the formant values are in Bark rather than Hertz. Adapted from De Boer (2001).*



	p	h	r
a	0	0	0
ɶ	0	0	1
ɐ	0.5	0	0
ɐ̹	0.5	0	1
ɑ	1	0	0
ɒ	1	0	1
e	0	0.5	0
ø	0	0.5	1
ɘ	0.5	0.5	0
ɵ	0.5	0.5	1
ɤ	1	0.5	0
o	1	0.5	1
i	0	1	0
y	0	1	1
ɨ	0.5	1	0
ʉ	0.5	1	1
ɯ	1	1	0
u	1	1	1

Figure 9: Different vowels in the [F_1, F'_2] space used for this simulation, with their values for position, height and rounding taken from De Boer (2001). Parameter values for the underlined vowels (open mid vowels) were not given in De Boer (2001); their approximate position in the vowel triangle has been drawn for better comparison with simulation results.

By calculating the [F_1, F'_2] outcome for the different possible combinations of these three parameters, two things may be achieved. First, a distinction can be made between vowels which are impossible to articulate (for instance, those with an F'_2 that is lower than its F_1) and those which are possible to articulate. Second, the [F_1, F'_2] values that may be articulated can be given a value representing their articulatory effort, by taking the average distance of each *articulatory* parameter from the neutral positions that result in a schwa. Different input values for the model are then mapped to the possible [F_1, F'_2] values in the OT grammars, and remembering the least effortful articulation for each [F_1, F'_2] pair, this effort can be translated to a ranking value for the articulatory constraint governing this particular pair. The assumption that the least effortful articulation resulting in an auditory value determines its effort value was also made by Ten Bosch (1991). Figure 10 gives the algorithm for calculating a ranking from the (p,h,r) values; Figure 11 visualizes the resulting rankings of the articulatory constraints in the grammar.

(a) For every possible combination of *p*, *h* and *r* from 0 to 1 in increments of 0.025, effort value E = $\frac{|0.5-p|+|0.5-h|+(0.5r)}{1.5}$, and calculate resulting [F_1,F'_2] values (rounded to the nearest 0.25 Bark and 0.5 Bark respectively).
(b) Find the lowest value E* for each [F_1,F'_2] pair in the grammar;
(c) If no value E can be found for a pair, ranking for this pair = 105
Else, ranking for this pair = 90+15E*

Figure 10: algorithm for determining the ranking value of different [F_1, F'_2] value pairs, based on the least effortful articulation resulting in that value (if any).

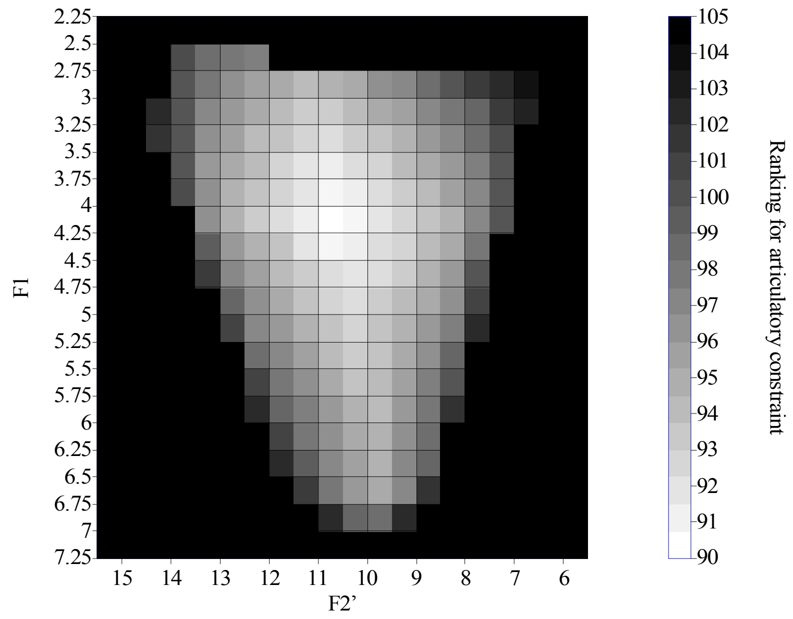


Figure 11: articulatory effort for the discretized [F_1, F'_2] value pairs that can be represented in the grammar. Darker shades of grey stand for articulations which require more effort; black areas are sounds which are very hard or impossible for human beings to produce.

4.2 Production, perception and learning in the OT grammar

Production

With the articulatory and cue constraints in place, a bidirectional OT grammar is formed which can map from underlying phonemes to surface [F_1, F'_2] values and vice versa, representing production and perception of vowels respectively. Figures 12 and 13 below exemplify these two processes. For the sake of clarity and brevity, these tableaus show only the constraints and candidates for a subset of the grammar, and only two possible phonemes /A/ and /B/.

Consider first the production tableau for phoneme /A/ in Figure 12. It is analogous to the production tableaus in Boersma & Hamann (2008), but with two-dimensional cue constraints (as in Boersma 2007). The 8 cue constraints all have a ranking value of 100.0, whereas the 4 articulatory constraints have different ranking values between 90.8 and 95.8, as assigned by the effort calculation formula in Figure 3. However, whenever the stochastic OT grammar evaluates a set of constraints, the rankings all are distorted with random *evaluation noise* drawn from a normal distribution with a mean of 0 and a standard deviation of 2. In this particular instance, this noisy re-ranking results in a constraint ranking where the articulatory constraint *[$F_1 = 5$, $F'_2 = 9$] is ranked higher than the cue constraint *[$F_1 = 5$] /A/, even though the latter has a higher true ranking value.

In this tableau, the winning candidate is /A/ [F_1=5, $F'_2 = 10$], meaning that in this particular instance the vowel phoneme /A/ will be realized with an F_1 of 5 Bark and an F'_2 of 10 Bark. For another production of this phoneme the outcome might very well be one of the other candidates: since the cue constraints pertaining to /A/ which partially determined the outcome all have the same true ranking value of 100.0, the order in which they appear in the grammar after noisy re-ranking is essentially random.

Crucially however, this does *not* mean that the true distribution of production outcomes for this grammar is completely uniform, because of the differing true ranking values of the articulatory constraints. The constraint *[$F_1 = 5$, $F'_2 = 9$], which also happens to violate a candidate in Figure 3, is on average more probable than the other articulatory constraints to be ranked highly after noisy re-ranking, because of its high true ranking value. This means that this constraint is also more likely determine the outcome of the evaluation process by eliminating a candidate. Conversely, the constraint *[F_1=4, F'_2=10] will on average have the least impact since it is likely to end up ranked low in the grammar due to its low true ranking value. In other words, if a very large number of production evaluations is performed with this particular grammar, it can be expected that the candidate which reflects the least effortful articulation will be produced most often.

Perception

Figure 13 gives an example of perception with the OT grammar, using the same noisy grammar that was used for production in Figure 12 and demonstrating cue interaction in perception (Escudero & Boersma 2004). This time around, there are two candidates, representing that the listener may perceive the incoming vowel with values [$F_1 = 5$, $F'_2 = 9$] either as the phoneme /A/ or /B/. The articulatory constraints are not violated by any candidates in the perception tableau, as these constraints apply only to production. On the other hand, cue constraints pertaining to /B/ *do* play a role since this phoneme is among the candidates.

The vowel phoneme /A/ wins in this tableau, because a cue constraint militating against one of the income values being perceived as /B/ is ranked higher than those forbidding that the incoming values are mapped to /A/. The fact that this ordering is purely random as the underlying (non-noisy) values for all cue constraints are all equal still applies. Since the articulatory constraints do not play a role in perception, the outcomes are actually equally likely; this means that if the evaluation process of perception were to be repeated a large number of times using this particular grammar, it is expected that the number of perceived of /A/ and /B/ tokens will be approximately equal, regardless of the formant values serving as input.

The grammar depicted in Figures 12 and 13 thus describes a language in which all vowels tend to be pronounced as a schwa (since it requires the least articulatory effort), and in which vowel sounds are perceived as arbitrary phonemes regardless of their quality. Such a language would of course be an extremely confusing and inefficient means of communication; nevertheless the agents in the simulation are 'born' with grammars of this type, where all cue constraints have the same true ranking. However, during their lifetime the agents are able to update their grammars to reflect the input they receive, as will be explained in the next section.

An important aspect about the production process is that the cue constraints may be violated by more than one candidate, as can be seen in Figure 12. This can result in a situation where all remaining candidates violate a constraint. In that case, the violations are not fatal and the candidates remain in the evaluation process. However since there is for each possible value pair a single articulatory constraint that forbids phonemes with those formant values, in the end only a single candidate will remain in the evaluation process; it cannot end in a 'draw'.

/A/	* [F'_2=10] /B/	* [F_1= 4] /B/	* [F_1= 4] /A/	* [F'_2=9] /B/	* [F_1= 5] /B/	* [F_1=5, F'_2=9]	* [F'_2=10] /A/	* [F'_2=9] /A/	* [F_1=5] /A/	* [F_1=4, F'_2=9]	* [F_1=5, F'_2=10]	* [F_1=4, F'_2=10]
	100	100	100	100	100	95.8	100	100	100	93	92.3	90.8
	101.8	*100.5*	*99.74*	*98.57*	*97.68*	*97.4*	*97.3*	*97.1*	*96.4*	*94.8*	*91.7*	*91.4*
[F_1=4, F'_2=9]			!*					*		*		
[F_1=4, F'_2=10]			!*				*					*
[F_1=5, F'_2=9]						!*		*	*			
☞[F_1=5, F'_2=10]							*		*		*	

Figure 12: production tableau for a token of phoneme /A/. The constraints are ranked according to their 'noisy' ranking, which is is written underneath their true ranking. Observe that the cue constraints pertaining to the phoneme /B/ do not play any role since only candidates representing phoneme /A/ are evaluated.

[F_1=5, F'_2=9]	* [F'_2=10] /B/	* [F_1= 4] /B/	* [F_1= 4] /A/	* [F'_2=9] /B/	* [F_1= 5] /B/	* [F_1=5, F'_2=9]	* [F'_2=10] /A/	* [F'_2=9] /A/	* [F_1=5] /A/	* [F_1=4, F'_2=9]	* [F_1=5, F'_2=10]	* [F_1=4, F'_2=10]
	100	100	100	100	100	95.8	100	100	100	93	92.3	90.8
	101.8	*100.5*	*99.74*	*98.57*	*97.68*	*97.4*	*97.3*	*97.1*	*96.4*	*94.8*	*91.7*	*91.4*
☞/A/								*	*			
/B/				!*	*							

Figure 13: perception tableau for a vowel with an F_1 of 5 Bark and an F'_2 of 9 Bark. The ranking values and the outcome of the stochastic re-ranking are the same as in Figure 12.

Updating cue constraints with the Gradual Learning Algorithm

In the simulation, it is assumed that when an agent hears another agent utter a vowel phoneme, it has contextual information besides the acoustic signal to determine which vowel was being uttered. This context may for example reside in the lexical and semantic level of the utterance, which are not modelled explicitly in the simulation. As a consequence, agents acting as listeners know which phoneme was intended by the speaker, and will know when they make an error in perception, i.e. when the winning phoneme of the perception process is not the same as the phoneme intended by the speaker (Escudero and Boersma 2004). Whenever this happens, agents will change the ranking values of the cue constraints responsible for the error by a small increment, using the GLA (Boersma 1997) that was described in Chapter 3.3.

Figure 14 gives an example of such a learning step taking place in an agent's grammar, as in Escudero and Boersma (2004). All constraints that were violated by the intended candidate are demoted by subtracting the value of the parameter PLASTICITY (this value is set to 0.02 in most of the simulations described here) from their ranking value. Likewise, all constraints that were violated by the erroneously winning candidate are promoted with this same value. These changes slightly decrease the likelihood of a /B/ with these [F_1, F'_2] values being misperceived as an /A/, although the effect of a single learning step is small in the face of the distortion that all rankings undergo at evaluation time. Over the course of an agent's life however, a large number of such learning steps takes place in its grammar, such that the ranking of the cue constraints comes to reflect the input this agent receives in its lifetime. Figure 15 visualizes how the rankings change over time for an agent that is fed a distribution of three vowels numbering approximately a million tokens per vowel.

[F_1=5, F'_2=9]	*[F'_2=10] /B/	*[F_1= 4] /B/	*[F_1= 4] /A/	*[F'_2=9] /B/	*[F_1= 5] /B/	*[F_1=5, F'_2=9]	*[F'_2=10] /A/	*[F'_2=9] /A/	*[F_1=5] /A/	*[F_1=4, F'_2=9]	*[F_1=5, F'_2=10]	*[F_1=4, F'_2=10]
	100	100	100	**99.98**	**99.98**	95.8	100	**100.02**	**100.02**	93	92.3	90.8
	101.8	*100.5*	*99.74*	*98.57*	*97.68*	*97.4*	*97.3*	*97.1*	*96.4*	*94.8*	*91.7*	*91.4*
☞ /A/								← *	← *			
✓ /B/				!* →	* →							

Figure 14: the same perception tableau as in Figure 7, but showing that the phoneme intended by the speaker was /B/ (indicated with a check mark). This perception error prompts the GLA to slightly change the true ranking values for the responsible constraints (the updated rankings are shown in bold type).

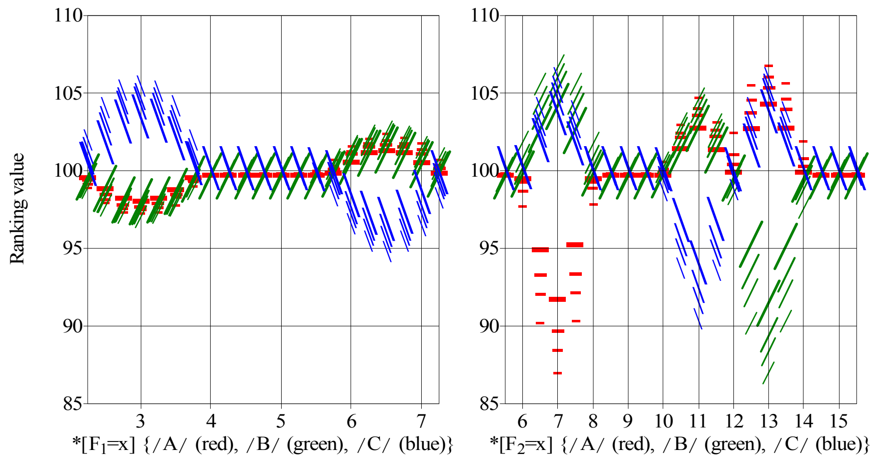


Figure 15. Ranking values for the F_1 (left) and F'_2 (right) cue constraints in the grammar of an agent over its lifetime as it is exposed to vowel tokens /A/ (minuses), /B/ (slashes) and /C/ (backslashes). The values for these input tokens are drawn from three normal distributions centered around [F_1, F'_2] value pairs of [3,7], [3,13] and [6.5,11] respectively, with a standard deviation of 0.5 Bark for both formants; the graph shows the rankings after 25 000 (thickest marks), 100 000, 250 000 and 1 000 000 (thinnest marks) timesteps. The evaluation noise was set at 2.0 and the plasticity at 0.01.

4.3 Interaction and population dynamics

Time and aging

The processes of speaking, listening and learning described in the previous section fully comprise the possible actions that can be undertaken by the agents in the grammar; it is through these three processes that a vowel system evolves when the agents interact in the simulation. Vowels 'spoken' by one agent may serve as input for another agent listening, and in the event this listener commits a perception error it will change its grammar. The simulation operates under the assumption that perception and learning precede production. To represent this, agents are given an 'age' measured in the number of iterations (timesteps) they have been present in the grammar; agents only start producing tokens that serve as perceptual inputs for other agents after they reach a certain age, which is defined by a parameter SPEAKINGAGE.

At regular intervals specified by the parameter BIRTHINTERVAL, a new agent is 'born' and added to the population, possessing a 'blank slate' grammar in which all cue constraints are set to an intial ranking of 100.0. To keep the population from growing out of control and to simulate the transmission of language from one generation to another, another parameter DYINGAGE specifies the age at which an agent will 'die' and be removed from the population. The size of the population can also be kept within bounds by specifying a maximum number of agents (encoded in a parameter MAXPOPULATIONSIZE) allowed to simultaneously exist in the simulation. These three variables determine whether the population size will grow, shrink or stagnate during the simulation.

It must be stressed that no 'reproduction' in the biological sense is simulated; all new agents are born in the simulation with identical grammars. This seems an appropriate assumption since this model investigates the (relatively) short-term *cultural* evolution of language, not the long-term *biological* evolution, and it is generally assumed that people will learn to speak the language of the environment they live in, completely irrespective of their genetic makeup[1].

Initialization

The simulation is initialized by placing a population of size STARTSIZE into the simulation world. All these 'newborn' agents have the true ranking values of their cue constraints set to 100.0 for all phonemes. The true ranking values of the articulatory constraints are set through the effort calculation algorithm given in Figure 10. In this starting situation, there are no 'older' agents yet from which these newborn agents may receive input tokens. To sidestep this chicken-egg problem, an initial input may be given to the agents, consisting of a number of tokens drawn from a distribution of [F_1, F'_2] values for each of these tokens. These values are distorted by the same transmission noise as the vowel tokens output by speaking agents (see next section). The initial input serves as a replacement for speaking agents during the time that no agents in the population have yet reached SPEAKINGAGE. A parameter INITIALINPUTDURATION determines the number of timesteps (starting from zero) that the initial input will be fed to the agents. For the results described in the next chapter, this variable was set to the same value as SPEAKINGAGE, so that as soon as a speaking agent surfaces in the population, the artificial input is put to a halt.

There is of course a danger that this initial input will have a large and defining influence on the language that the agents in the simulation end up speaking. An earlier version of the simulations described here, in which a one-dimensional sibilant contrast was simulated (van Leussen & Vondenhoff 2008), showed that for this contrast the agents reached a stable 'optimal' state whether initial inputs were given or not, although it took longer for the population to reach this state. The effect of giving an initial input will therefore also be examined in the next chapter. A parameter GIVEINITIALINPUT, which can be either true or false, determines whether initial input will be given to the agents at all.

Transmission and interaction

In actual spoken language use, words and the phonemes they contain are not directly transmitted from the brain of the speaker to that of the listener. Rather, they are transmitted as sounds originating from the vocal tract of the speaker, propagating through the air and arriving at the auditory system of the listener. The intended sound may be subject to different forms of noise at all these levels, such as variations in the muscle control of the speaker, in background noise, and in the ear of the listener

[1] Although see Dediu & Ladd (2007), which describes a possible causal relation between certain genetic traits and the typological linguistic feature of tone.

(Boersma & Hamann 2008). Indeed it is thought that these distortions may contribute substantially to language change and variation (e.g. Blevins 2004, Ohala 1981). In the simulation described here, articulatory variation and background noise are collapsed into a *transmission noise* variable, as in Boersma & Hamann (2008). However as my simulation deals with two different dimensions that serve as phonological cues, the noise affecting these cues is encoded in two different variables TRANSMISSIONNOISEF_1 and TRANSMISSIONNOISEF_2.

The effect of these variables is to add a value drawn from a normal distribution with a mean of zero and a standard deviation equal to the value of TRANSMISSIONNOISEF_x to the [F_1, F'_2] values of a speaker's outputs before it is processed by listening agents. Because the distributions are continuous whereas the values that can be represented by the grammar are discrete, the resulting formant quantities are then rounded to the nearest F_1 and F'_2 values that can be represented by the constraints of the grammar. Similarly, values that lie *outside* the value bounds present in the grammar (for example if an F_1 value exceeds its upper bound of 7.25 Bark after adding noise) will be rounded to the boundary that they exceeded. As this last type of rounding possibly biases the grammar toward these boundary values, the possible space of values for the grammar has been chosen such that values lying outside this space will be generated very rarely under transmission noise values deemed reasonable. The values that result from this distorting and rounding are fed as input to the agents serving as listeners. Figures 16ab show the effects of adding transmission noise to an output value pair, and of discretization of these distorted values.

Interaction in the simulation takes place as follows. At each timestep, every agent that is of SPEAKINGAGE produces a single output token for a randomly chosen phoneme, through the OT grammar production process described above. This output token is distorted with transmission noise and then added to list a inputs for all *other* agents. Agents do not process their own outputs, and the distorted values are the same for all listening agents. After this "speaking phase", all agents which have received input tokens process these through their OT grammar, and update their constraint rankings if misperception occurs. These speaking and listening/learning phases are repeated throughout the simulation. In this way, a common vowel 'language' is established and maintained through generations of agents, while the population and its size changes due to the population control variables described above.

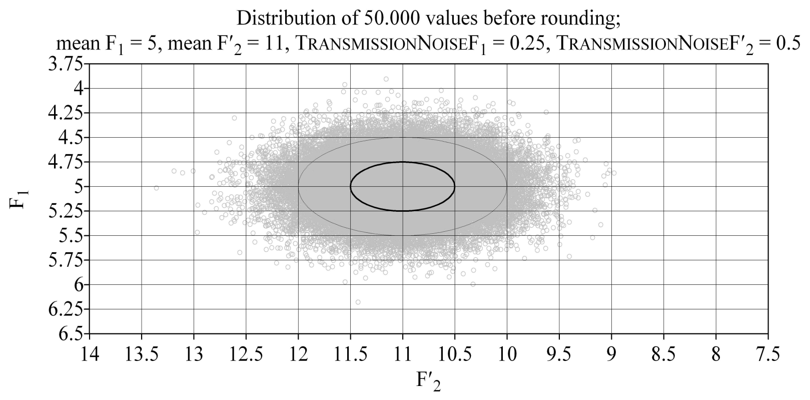


Figure 16a: a scatterplot of 50.000 outputs with [F_1=5, F'_2=11] after adding transmission noise.
The solid ellipse represents one standard deviation from the mean, the dotted ellipse two standard deviations.

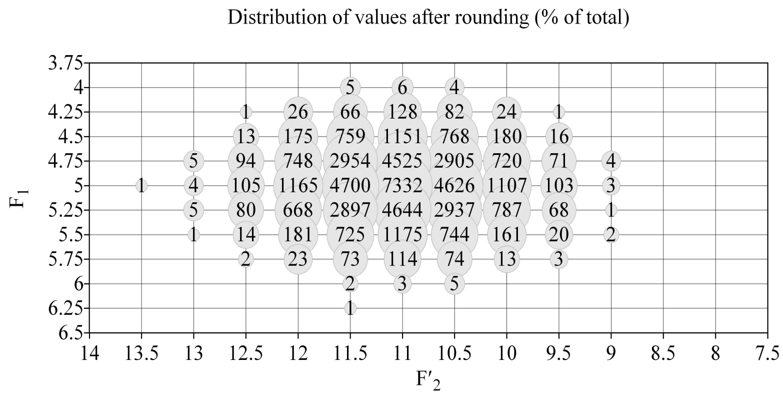


Figure 16b: distribution of the values from figure 16a after discretization. Only 7332, or 14.6%, of value pairs are identical to the original output values.

4.4 Overview

The above sections describe all aspects of the simulation algorithm used to generate the results described in the next chapter. Figure 17 gives a summary and step-by-step overview of the algorithm in pseudocode. The actual Java implementation can be found in Appendix I.

```
Main loop
Initialization variables
a set ArticulatoryConstraints[] with rankings according to effort (see Figure 10);
a set of initialized agents agents[] with size startSize;
a set of phonemes[]  {A,B,..}
numberOfTimesteps; birthInterval; dyingAge; speakingAge; maxPopulationSize;
a parameter giveInitialInput which is either true or false; an optional set of [F1 ,F'2] initialValues[] for the phonemes, and a
duration initialInputDuration for this input.
for timesteps = 0 to numberOfTimesteps
    if (timesteps mod birthInterval = 0 and size of agents[] < maxPopulationSize) then
        newAgent = INITIALIZE_NEW_AGENT
        add newAgent to agents[]
if (timesteps < initialInputDuration)                     }  This part of the code is only
            for every phoneme in phonemes[]                }  executed when initial input is
                        Candidate input ← CREATE_INITIAL_INPUT }  turned on in the simulation, i.e.
                        for every listeningAgent in agents[]   }  when giveInitialInput is true.
                                    add output to listeningAgent.tokens[]
    for every agent in agents[]
        agent.age = agent.age + 1
        if agent.age ≥ speakingAge then
                candidate output ← agent.SPEAK
                for every listeningAgent in agents[]
                        if listeningAgent ≠ speaking agent
                        add output to listeningAgent.tokens[]
        if agent.age ≥ dyingAge then
                remove agent from agents[]
    for every agent in agents[]
        for every candidate in agent.tokens[]
                agent.PROCESS(candidate)
        clear agent.tokens[]
end
```

(continued on next page)

```
agent.SPEAK
Randomly select a phoneme from phonemes[]
for F1value from 2.25 to 7.25 in steps of 0.25
    for F'2value from 5.5 to 15.5 in steps of 0.5
        add candidate(/phoneme/,[F1value, F'2value]) to candidates[]
Candidate winner ← agent.DETERMINE_WINNER(candidates[])
Candidate winner. F1value += Gaussian noise ($\mu = 0$ , $\sigma$ = TRANSMISSIONNOISE$F_1$ ))
Candidate winner. F'2value += Gaussian noise ($\mu = 0$ , $\sigma$ = TRANSMISSIONNOISE$F'_2$))
Candidate output ← round to nearest possible formant values(winner)
clear candidates[]
return output
```

```
agent.PROCESS(Candidate input)
for every phoneme in phonemes[]
    add Candidate(phoneme, input.value) to candidates[]
Candidate winner ← agent.DETERMINE_WINNER(candidates[])

if winner.phoneme ≠ input.phoneme then
    promote Constraint "*perceived$F_1$ Value winner.phoneme" by adding plasticity to trueRankingValue
    demote Constraint "*perceived$F_1$ Value input.phoneme" by subtracting plasticity from trueRankingValue
    promote Constraint "*perceived$F'_2$Value winner.phoneme" by adding plasticity to trueRankingValue
    demote Constraint "*perceived$F'_2$Value winner.phoneme" by subtracting plasticity from trueRankingValue
```

```
agent.DETERMINE_WINNER(candidates[]) returns candidate
for every Constraint in Grammar
    Constraint.disharmony = Constraint.trueRankingValue+Gaussian noise($\mu = 0$, $\sigma$ = EvaluationNoise)
while size of candidates[] > 1
    for every constraint in Grammar (ordered by disharmony)
        for every candidate in candidates[]
            if candidate violates constraint then
                add candidate to violators[]
        if size of violators[] < size of candidates[] then
            remove every Candidate in violators[] from candidates[]
return last candidate in candidates[]
```

```
INITIALIZE_NEW_AGENT returns agent
create newAgent
newAgent.age = 0
add ArticulatoryConstraints[] to newAgent.Grammar
for F1value from 2.25 to 7.25 in steps of 0.25
    for F'2value from 5.5 to 15.5 in steps of 0.5
        for each phoneme in phonemes[]
            add (cue) Constraint(*[F1value, F'2value] /phoneme/) to newAgent.Grammar,
              with trueRanking=100
return newAgent
```

```
CREATE_INITIAL_INPUT returns candidate
Randomly select a phoneme from phonemes[]
Create a candidate input with /phoneme/, and F1value and F2value according to initialValues[phoneme];
Candidate input. F1value += Gaussian noise ($\mu = 0$ , $\sigma$ = TRANSMISSIONNOISE$F_1$ ))
Candidate input. F'2value += Gaussian noise ($\mu = 0$ , $\sigma$ = TRANSMISSIONNOISE$F'_2$))
Candidate input ← round to nearest possible formant values(input)
```

Figure 17: Pseudocode for the algorithm used in the simulation.

5 Results

In this chapter, the outcome of running the simulations under different parameter settings is shown. In the first section, the effect and necessity of the TRANSMISSIONNOISEFx and EVALUATIONNOISE parameters is shown. Next, section 5.2 will show the vowel systems predicted by the model for 3 to up to 7 phonemes. In section 5.3 the effect of the population parameters is explored, and 5.4 looks into the ability of the model to account for the diachronic phenomena of mergers, splits and (chain) shifts.

The results of the simulations will be compared to real vowel systems in a somewhat impressionistic fashion; that is, the systematic numerical comparison between the vowel systems of the simulations and that of actual languages found in De Boer (2001) and Schwartz et al. (1997) will not be imitated here, as I believe the investigative and tentative experiments described in this thesis do not yet justify such analysis – a cursory glance will reveal that the prediction of vowel qualities is flawed at times.

5.1 Effects of the noise parameters

It has been mentioned in chapter 5 that the perception, production and learning processes are subject to noise at two different levels: outside the grammar values may be distorted by the TRANSMISSIONNOISEF_x variables, and during evaluation the grammar itself is subject to EVALUATIONNOISE. Because injecting noise into the simulation two times may seem superfluous, I will first illustrate the distinct effects of these two sources of noise on the outcome of the simulations with 3 phonemes /A/, /B/ and /C/.

Transmission noise

The phonemes were given initial [F_1, F'_2] values of [4,8], [4,12] and [5,10]. Population effects are not simulated yet here: STARTSIZE is 1, BIRTHINTERVAL is set to 10,000, SPEAKINGAGE is set to 10,000 and DYINGAGE is set to 20,000. Effectively this means that after the initialization phase, there are always two agents in the population: a speaking agent and a learning agent. The evaluation noise was set to 2.0 and the plasticity to 0.1. All these simulations ran for a total of 100,000 timesteps. Figures 18abc compare the outcome of experiments with no transmission noise, a small amount of transmission noise, and a large amount of transmission noise.

It can be seen that the transmission noise variable is critical to the behavior of the simulation. If it is turned off, as in Figure 18a, transmission of phonemes from speaker to learner becomes rather erratic, with strange fluctuations in the quality and varations of the vowels. Setting the transmission noise to unrealistically high values also results in unrealistic behavior, as in Figure 18c where it is set to 2 Bark for both first and effective second formant. These high values severely distort the contrast between the phonemes, rendering the language unlearnable. Finally, Figure 18b shows the outcome for an F_1 transmission noise of 0.25 Bark and an F'_2 transmission noise of 0.75 Bark. With these values, the agents will learn the language they receive as input, but dispersion also occurs over time, moving the phonemes closer to the periphery of the vowel space.

In the rest of the simulations described in this chapter, the transmission noise variables will be set to 0.25 Bark for F_1 and 0.75 Bark for F'_2 unless stated otherwise.

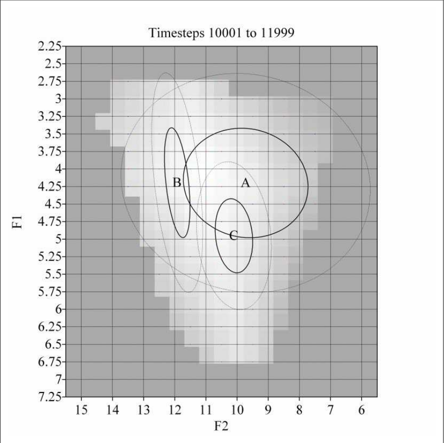
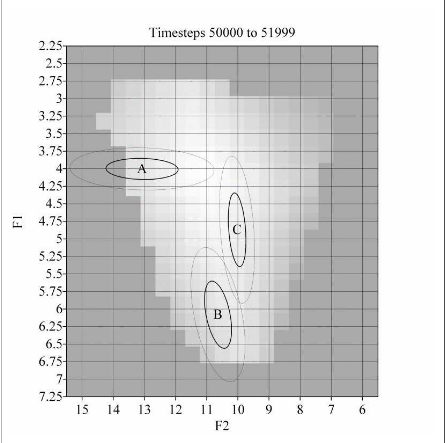
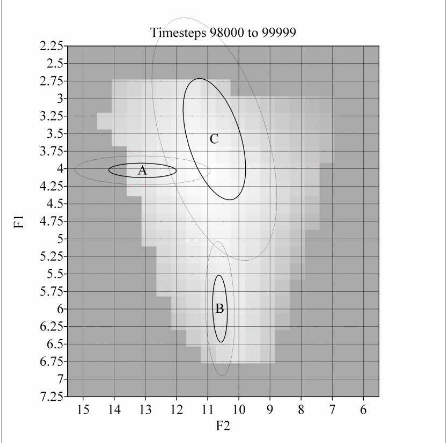
TransmissionNoiseF_1 *=0, TransmissionNoise*F'_2 *=0*		
		
Outputs at timesteps 10,000-11,999	Outputs at timesteps 50,000 - 51,999	Outputs at timesteps 98,000-99,999

Figure 18a: Scatterplots of the output of all agents at three different intervals in a simulation with no transmission noise. The solid and dotted ellipses around the phoneme labels represent one and two standard deviations from the mean. Because there transmission noise is zero, values are mostly plotted on top of each other at discrete points in the grid.

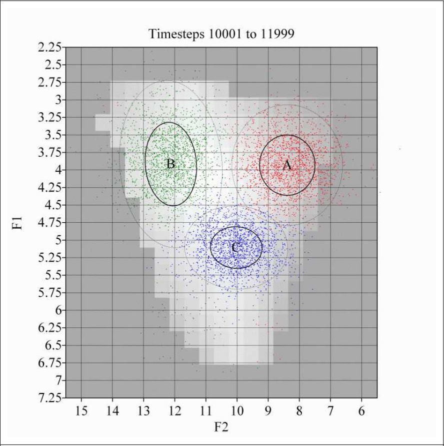
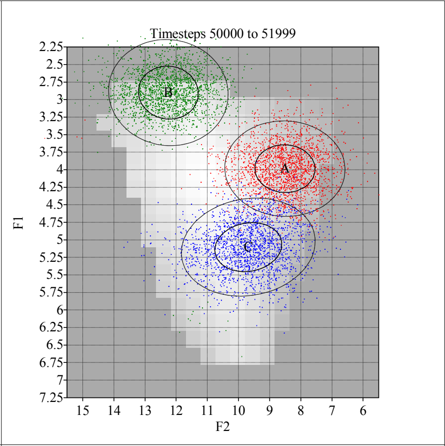
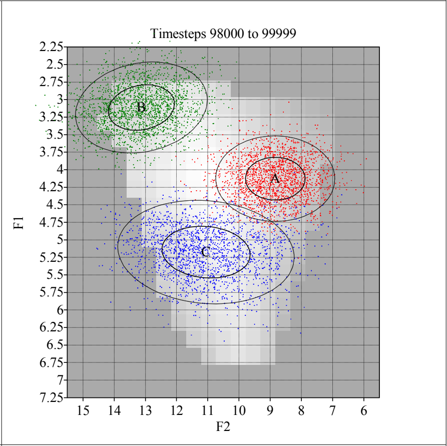
TransmissionNoiseF_1 *= 0.25, TransmissionNoise*F'_2 *=0.75*		
		
Outputs at timesteps 10,000-11,999	Outputs at timesteps 50,000 - 51,999	Outputs at timesteps 98,000-99,999

Figure 18b: as 18a, but for a simulation with transmission noise values of 0.25 and 0.75 for the first and effective second formant. Dispersion occurs and remains stable over the generations.

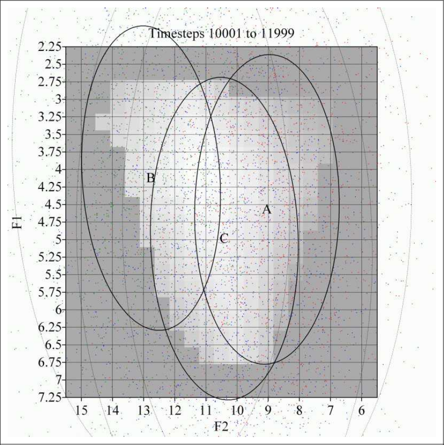
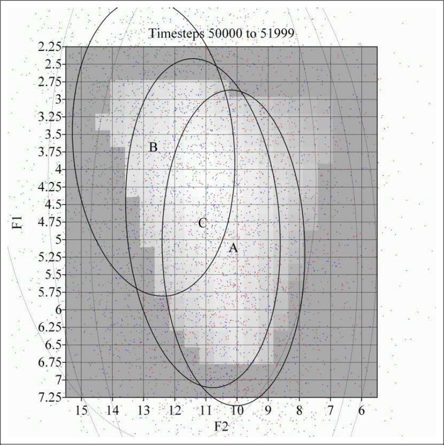
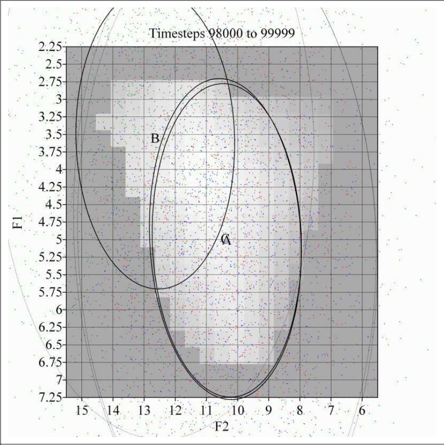
TransmissionNoiseF_1 *=2, TransmissionNoise*F'_2 *=2*		
		
Outputs at timesteps 10,000-11,999	Outputs at timesteps 50,000 - 51,999	Outputs at timesteps 98,000-99,999

Figure 18c: as 18a and 18b, but with high transmission noise values. A dispersed vowel system does not emerge under these parameter settings.

Evaluation noise

Chapter 3.2 discussed evaluation noise as a necessary parameter to model speaker-internal variation in stochastic OT grammars (Boersma 1997). Boersma & Hamann (2008) consistently use an evaluation noise value of 2.0, which was also used for the simulations described below. The DYINGAGE and BIRTHINTERVAL parameters were again set at 20,000 and 10,000 respectively; SPEAKINGAGE was 10,000, only one agent was put into the simulation at the start, and again the experiments ran for a total of 100,000 timesteps. The transmission noise values were now kept constant at TRANSMISSIONNOISEF_1 $= 0.25$, TRANSMISSIONNOISEF'_2 $= 0.75$. Figure 19a shows the effect of turning off evaluation noise on the outputs of two agents during the course of the simulation; Figure 19b shows the effect of setting it to a high value of 10.0. Vowel dispersion effects still occur with these settings, and the last agent in the simulation ended up with a vowel system close to /a/, /e/, /o/; however the effects are somewhat erratic and the vowel systems do not reach a 'stable state' over time as they do for the standard evaluation noise setting of 2.0.

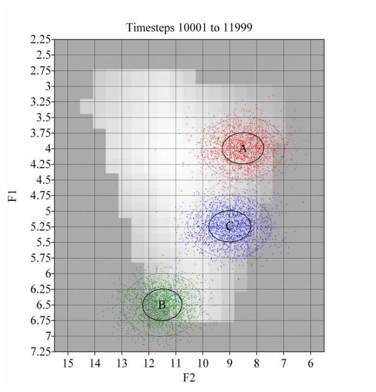
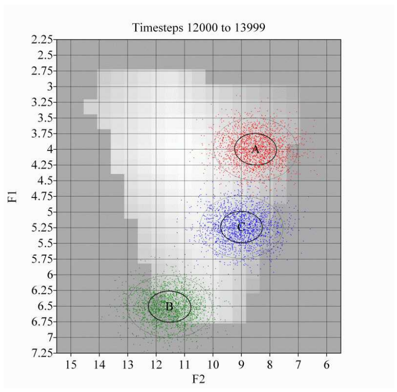
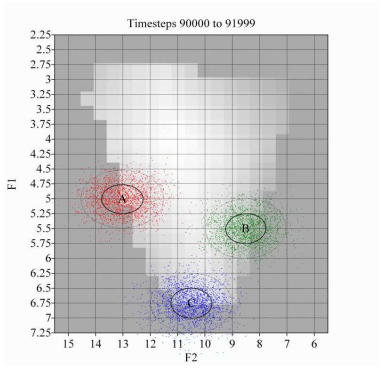
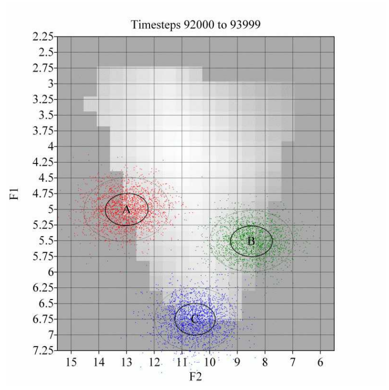
Evaluation noise = 0	
	
Output of Agent 0 (first speaking agent in the simulation) at timesteps 10,000-11,999	*Output of Agent 0 at timesteps 12,000-13,999*
	
Output of Agent 9 (last speaking agent in the simulation) at timesteps 90,000-91,999	*Output of Agent 9 at timesteps 92,000-93,999*

Figure 19a. Outputs of the first and last speaking agents in the simulation with no evaluation noise. The output of the grammars is completely invariant; the deviations are caused by the transmission noise (0.25 for F_1 *and 0.75 for* F'_2).

Setting evaluation noise to 2.0 and transmission noise to 0.25 for F_1 and 0.75 for F'_{2},, as shown in Figure 18b, seems to show the most realistic effects, both in the change of the agents' outputs over time and in the resulting vowel systems. Therefore the next section will keep these variables constant and look into the vowel systems generated by the agents under these settings.

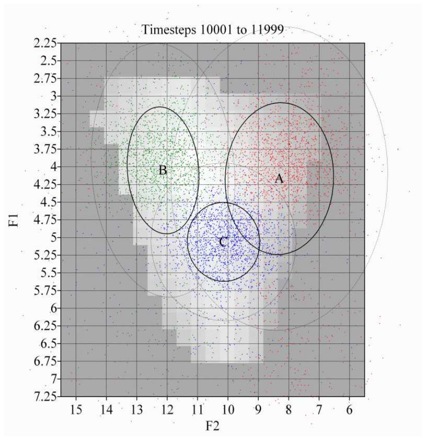
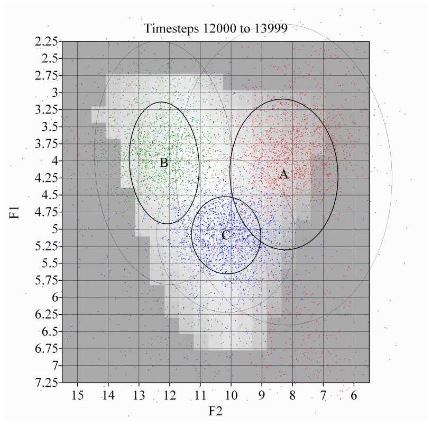
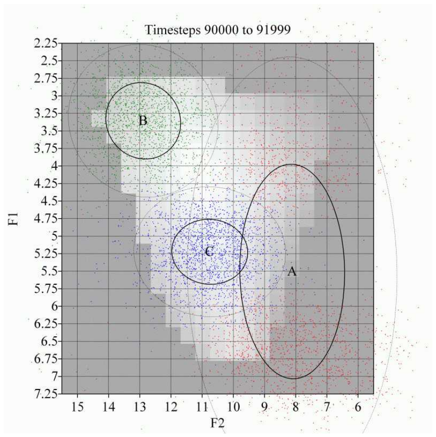
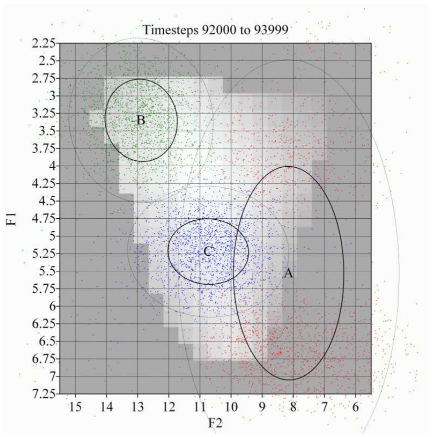
Evaluation noise = 10	
	
Output of Agent 0 at timesteps 10,000-11,999	*Output of Agent 0 at timesteps 12,000-13,999*
	
Output of Agent 9 at timesteps 90,000-91,999	*Output of Agent 9 at timesteps 92,000-93,999*

Figure 19b. Outputs of the first and last speaking agents in the simulation with the evaluation noise set to 10. The high evaluation noise results in erratic learning behavior and unrealistic vowel dispersions (for instance the values for vowel /A/ in both agents lie largely outside of the possible vowel space).

5.2 Optimal vowel systems predicted by the model

This section looks into the effects of changing the number of vowels the agents have in their phonemic repertoire. To make sure that they were not predisposed toward particular vowel configurations, no initial input was given to the agents in these simulations (the shaded part of the code in Figure 17 was not executed). Figures 21a-e show the results. For comparison to real vowel systems, Figure 20 reproduces the estimated $[F_1, F'_2]$ values for different phonemes from Figure 9. The evaluation noise was set 2.0 and the transmission noise to 0.25 for F_1 and 0.75 for F'_2 in all these simulations.

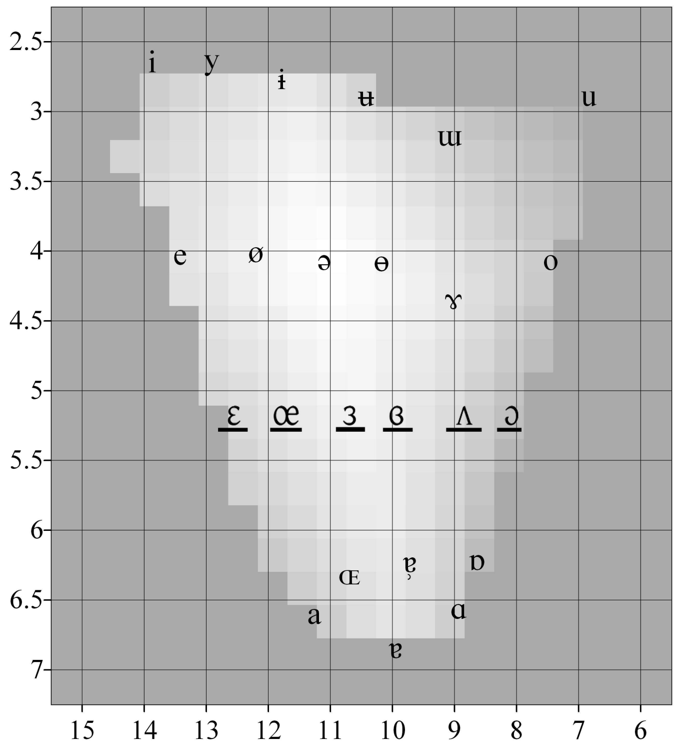


Figure 20. As Figure 9, but with the symbols for different phonemes superimposed on a graph encoding relative articulatory effort, as the results in Figures 21a-e

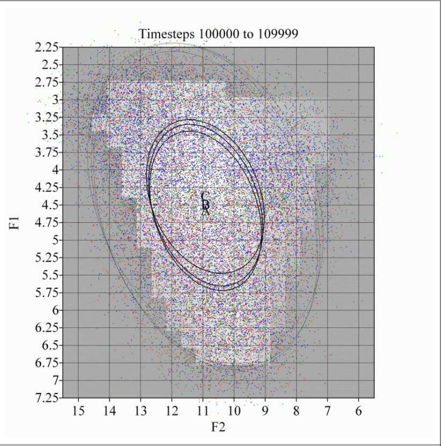
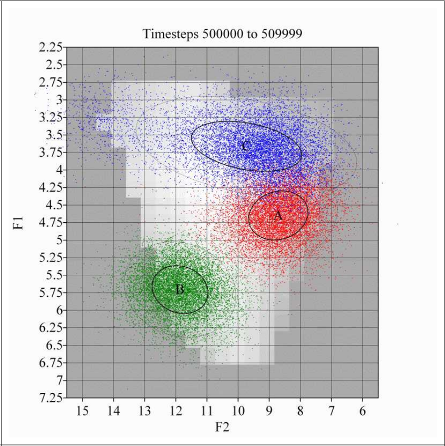
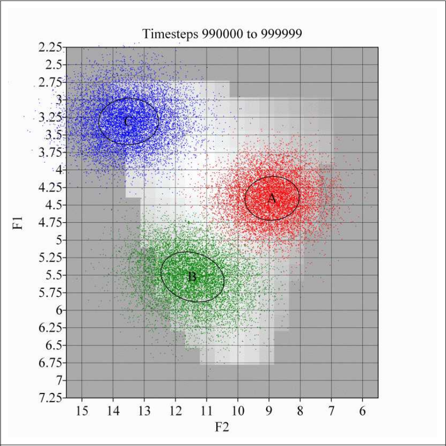
		
Agents' outputs at timesteps 100,000 to 109,999	Agents' outputs at timesteps 500,000 to 509,999	Agents' outputs at timesteps 1,000,000 to 1,009,999

Figure 21a: Outcome for a 3-vowel simulation. The agents reach a stable state with a vowel system resembling /u/, /ɛ/, /i/ or perhaps /o/, /ɛ/, /i/.

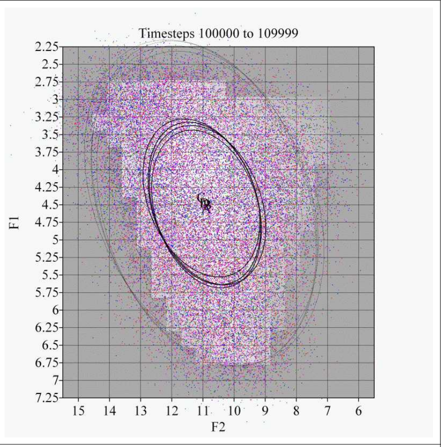
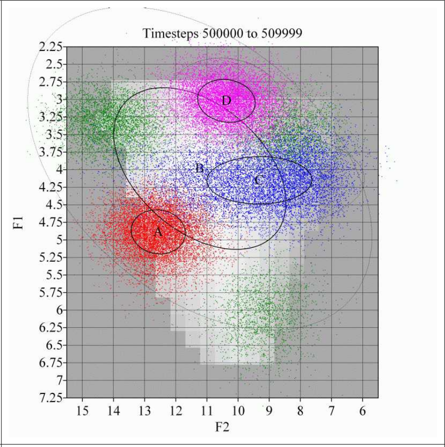
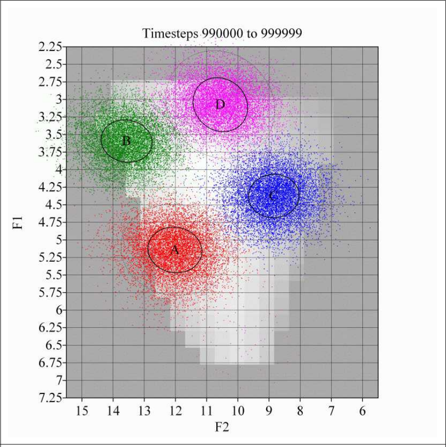
		
Agents' outputs at timesteps 100,000 to 109,999	Agents' outputs at timesteps 500,000 to 509,999	Agents' outputs at timesteps 1,000,000 to 1,009,999

Figure 21b: Outcome for a 4-vowel simulation. The agents reach a stable state with a vowel system resembling /u/, /ɛ/, /i/, /ɨ/.

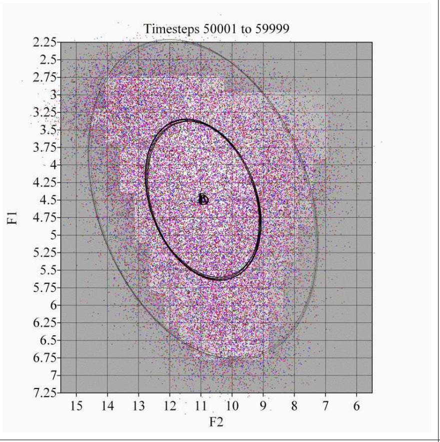
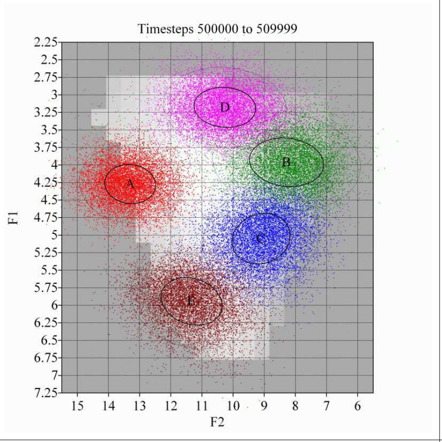
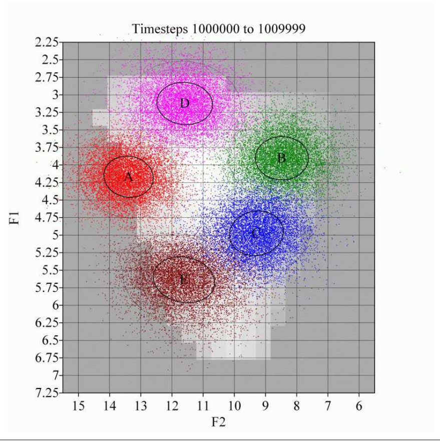
		
Agents' outputs at timesteps 100,000 to 109,999	Agents' outputs at timesteps 500,000 to 509,999	Agents' outputs at timesteps 1,000,000 to 1,009,999

Figure 21c: Outcome for a 5-vowel simulation The agents reach a stable state with a vowel system resembling /u/, /ɛ/, /i/, /ɨ/, /o/.

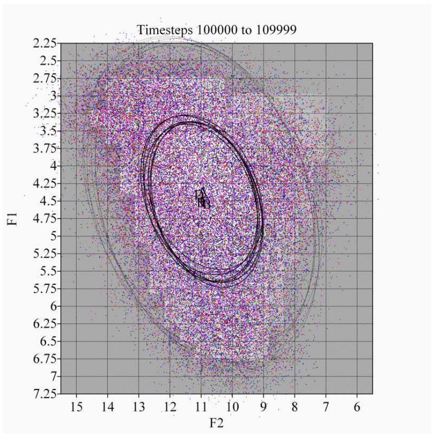
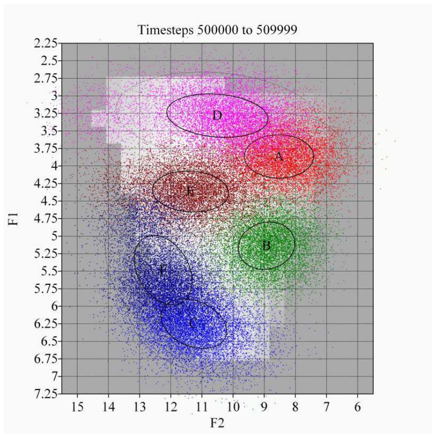
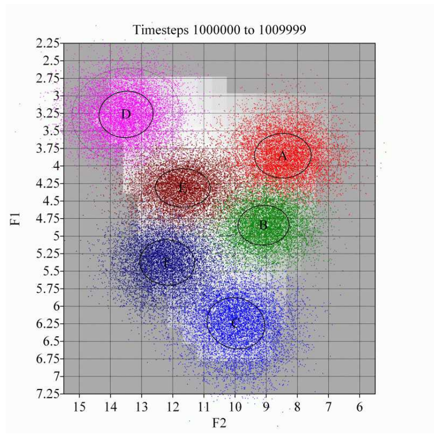
		
Agents' outputs at timesteps 100,000 to 109,999	Agents' outputs at timesteps 500,000 to 509,999	Agents' outputs at timesteps 1,000,000 to 1,009,999

Figure 21d: Outcome for a 6-vowel simulation. The agents reach a stable state with a vowel system resembling /u/, /ɛ/, /i/, /ɪ/, /o/, /a/.

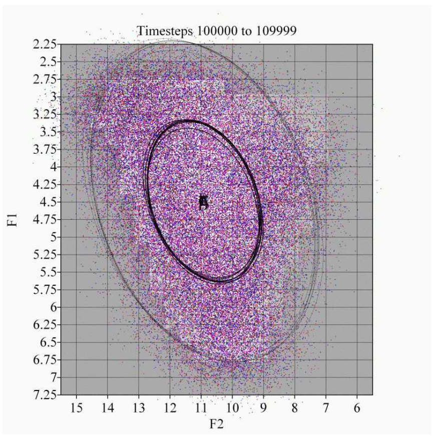
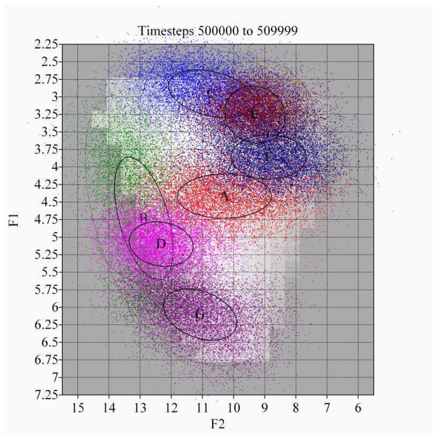
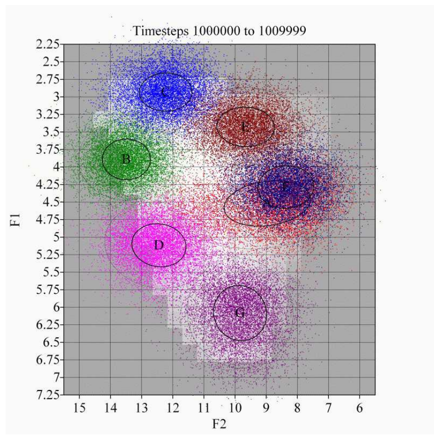
		
Agents' outputs at timesteps 100,000 to 109,999	Agents' outputs at timesteps 500,000 to 509,999	Agents' outputs at timesteps 1,000,000 to 1,009,999

Figure 21e: Outcome for a 7-vowel simulation. Interestingly, full dispersion does not occur for this number of vowels: the phonemes with labels A and F fail to disperse and overlap significantly. The resulting system can be described as /ɯ/, /ɛ/, /i/, /ɨ/, /o/, /o/, /a/.

A quick glance at the results confirms that the model is reasonably succesful at predicting the emergence of auditorily dispersed vowel systems. For 3 to 6 vowels, the artifical language goes from a state of complete arbitrariness and confusion to a well-ordered system of non-overlapping vowels. The inefficient language that the agents start out with disappears relatively quickly; after that, movement toward a dispersed system occurs at an ever-declining pace until directional change no longer occurs and a more or less stable system is reached.

An interesting effect that occurs during the transition to a stable system is that some phonemes are multimodally distributed: that is, they are realized not as a single cluster, but as two or more clusters with other phonemes in between. Since in these simulations at most one agent is producing outputs at any given moment due to the settings of the BIRTHINTERVAL, SPEAKINGAGE and DYINGAGE parameters, they are not the result of inter-speaker differences. However the multimodal distributions are harder to learn for the listening agents and sooner or later one of the clusters will gain the upper hand in the learning process until the distribution is effectively monomodal. While I am not aware of languages that allow for split distributions in the vowel system of a single speaker, Boersma & Hamann (2008) also simulate multimodal phonemes for the case of sibilant inventories and speculate that Dutch may be developing such a split. Incidentally, these split distributions are not possible in exemplar models of phoneme acquisition like Wedel (2006).

For 7 vowels however, complete dispersion did not occur within the timeframe of the simulation: two phonemes have significantly overlapping distributions. The model thus predicts that in the case of 7 vowels and under these specific parameters, an optimally dispersed vowel system based only on F_1 and F'_2 cue distinctions is not possible. This prediction is not accurate, as 7-vowel systems with no secondary distinctions (for example length or phonation type) are attested in the UPSID database by Schwartz et al. (1997a). They do find a tendency for systems with more vowels to use secondary distinctions, but state that this trend starts to occur for systems with 9 vowels.

It would be tempting to call the overlapping vowels in Figure 21e a case of a vowel merger, and say that the simulation can account for this diachronic phenomenon. However to do so would be inaccurate. The architecture of the simulation does not allow agents to 'merge' from a 7-vowel system to a 6-vowel system, so the overlapping vowels are still considered two phonemes by the agents. Thus, here a language is simulated where two vowel phonemes are indistinguishable from one another and cause perpetual confusion among listeners since the difference seems to be unlearnable.

While the model is mostly able to predict the development of vowel dispersion attested in real languages, the vowel systems that it predicts are not always quite accurate in terms of vowel quality. Figure 1 in Chapter 2 shows systems with /a/, /i/, /u/ to be the most common by far. The model does predict the emergence of /i/ and /u/-like phonemes, but the vowel /ɛ/ is often found where an /a/ is typologically preferred. Likewise, for $n = 5$ the model predicts something like /u/, /ɛ/, /i/, /ɨ/, /o/, containing the rare /ɨ/ instead of the unmarked /a/. Generally the model seems too inclined towards central close vowels and toward low F_1 values. The implications of this will be discussed in the next chapter.

5.3 Population effects

So far, results have only been shown for simulations in which no more than a single speaker and no more than a single learner existed in the simulation world simultaneously. However the model is also able to handle more complex population dynamics (Chapter 4.3). Figure 22 shows the emergence of a vowel system in a growing population, with the population control parameters set to the following values: MAXPOPULATIONSIZE = 5, BIRTHINTERVAL = 5000, SPEAKINGAGE = 10,000, DYINGAGE = 30000. The simulation ran for a total of 200,000 timesteps, so a total of 40 agents was 'born' during the simulation. As in the previous section, no initial input was given. This means that in contrast to the runs described previously, agents still changed their vowel systems after they started producing tokens themselves, and received inputs from different sources.

Under these settings dispersion still occurs, but the resulting vowel system differs in a number of ways from those predicted in the previous section. There seems to be a preference for higher F_1 and F'_2 values compared to the runs with single learners, and the vowels are less dispersed but driven more toward the periphery of the vowel space, so that the resulting system contains four front vowels.

In De Boer (2001), increasing the population size mainly had the effect of increasing the stability of the resulting vowel systems. It would be interesting to further examine the effects of different population sizes and dynamics in the model described in this thesis, especially by introducing inter-agent differences. Unfortunately, due to time constraints (increasing the population size dramatically increases the computational complexity of the model), these possiblities will only be discussed as conjectures in the next chapter.

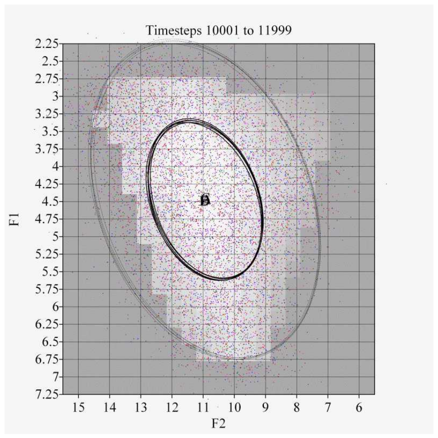
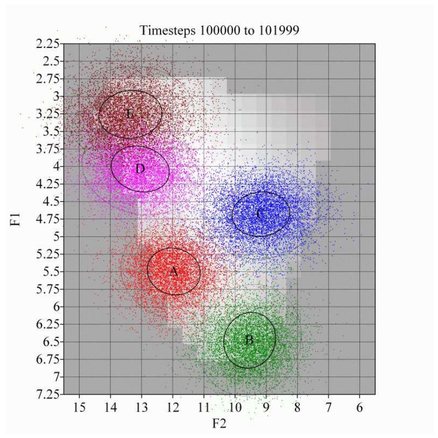
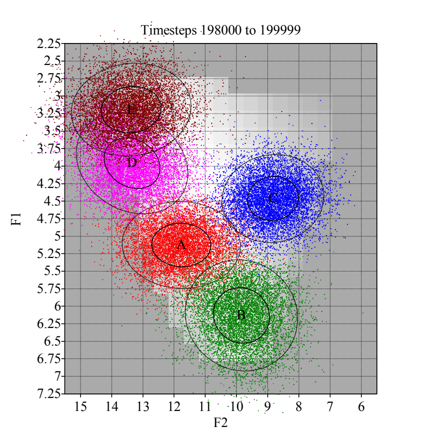
		
Outputs of all agents at timesteps 10,000 to 11,999	Outputs of all agents at timesteps 100,000 to 101,999	Outputs of all agents at timesteps 198,000 to 199,999

Figure 22: Vowel dispersion in a population able to grow to the size of 5 agents. While dispersion still occurs, there are some differences with the 5-vowel system predicted in 5.2; most notably, /a/ does appear while /u/ does not. The resulting vowel system is approximately /i/,/ɪ/, /ɛ/,/a/,/o/.

5.4 Mergers, splits, and shifts?

Since the number of phonemes available to the agents is static during the course of the simulation, and due to the abstractions made from actual language use, the diachronic processes of vowel mergers and splits cannot and should not be modelled in the simulation as described in the previous chapter. However as in Boersma & Hamann (2008), it *is* possible to consider situation in which a merger or split has just occurred, resulting in a non-optimal dispersion of the vowel system. Boersma & Hamann's model predicted that chain shifts occur in such a situation: all vowels in the system shift to adapt to the new size of the inventory.

To imitate vowel systems in which a merger has just occurred, a run of the simulation was made in which the stable five-vowel outcome shown in Figure 21c was fed as initial input to the agents with one vowel removed, creating a void. Likewise, a split was imitated by taking this output and giving it as input in a new run, but with an extra phoneme at the same spot in the vowel space as one of the original five vowels (Figures 23ab). Figures 24 and 25 show the resulting adaptations. The simulation ran for 200,000 timesteps, with DYINGAGE = 20,000 and BIRTHINTERVAL = 10,000, so that a total of 20 'generations' was imitated.

Both situations indeed resulted in vowel shifts. The 'merger' shows that the newly created vacuum is quickly filled up by the 'merged' vowel. In reaction, the other vowels also take on a new position in the vowel space, pushing each other along in a clockwise motion. The 'split' also causes a vowel shift, but not convincingly for all vowels, and not all in the same direction. Strangely, the newly created vowel (labeled B in Figure 24) seemingly 'skips over' another vowel to fill the vacuum found there, as sometimes occurred as a result of the multimodal distributions discussed in section 5.2.

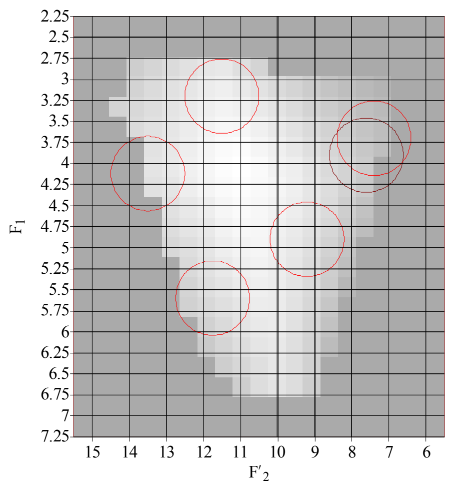


Figure 23a: a simulated split based on Figure 21c (the phoneme drawn as a dotted circle was added)

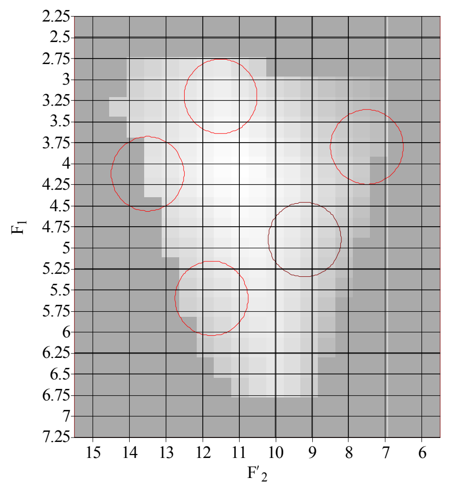


Figure 23b: a simulated merger based on Figure 21c (the phoneme drawn as a dotted circle was removed)

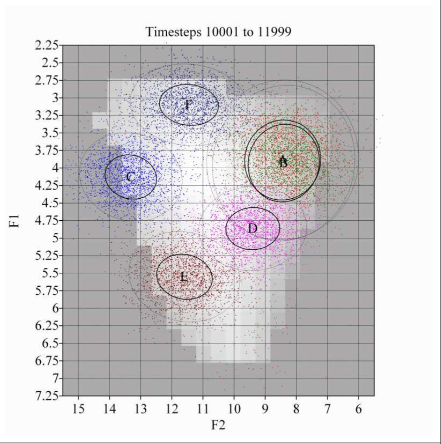
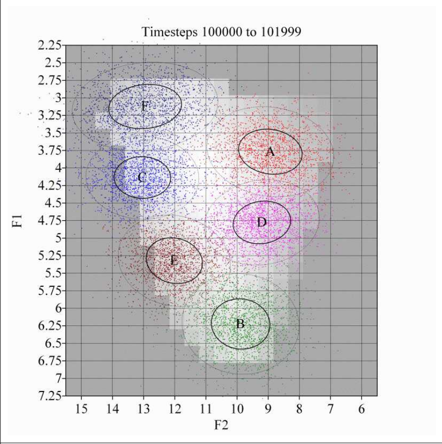
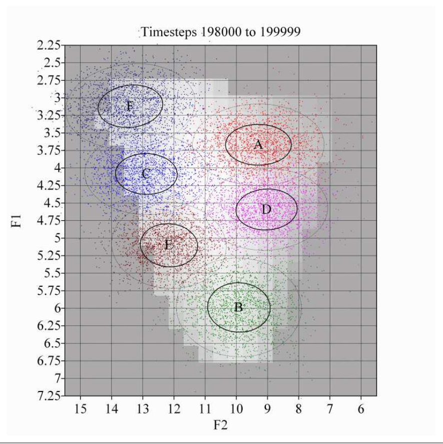
		
Outputs of all agents at timesteps 10,000 to 11,999	*Outputs of all agents at timesteps 100,000 to 101,999*	*Outputs of all agents at timesteps 198,000 to 199,999*

Figure 24a: the results of giving an input in which an artificial vowel split has taken place. Some vowels shift, others do not.

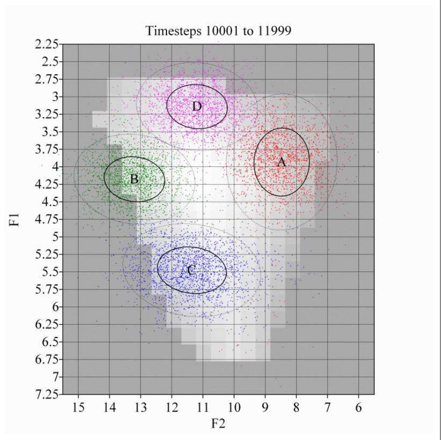
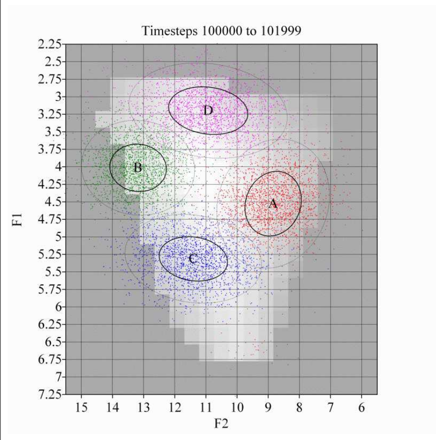
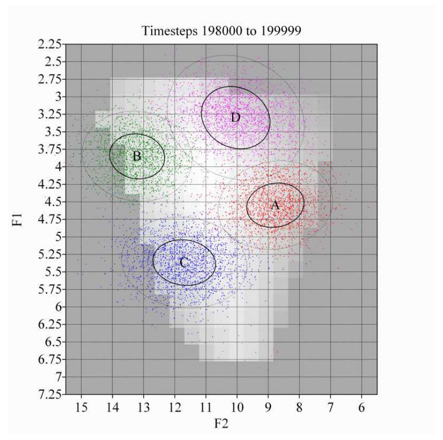
		
Outputs of all agents at timesteps 10,000 to 11,999	*Outputs of all agents at timesteps 100,000 to 101,999*	*Outputs of all agents at timesteps 198,000 to 199,999*

Figure 25: the results of giving an initial input in which an artificial vowel merger has taken place. Here a convincing chain shift occurs: the phoneme labeled A moves into the spot vacated through the merger, and the other vowels shift along in clockwise direction (most notably D).

6 Discussion and conclusion

6.1 Interpretation of the results

This thesis has presented some results of computer-aided simulation experiments aiming to integrate phonetically-based models of vowel dispersion into the framework of Stochastic Optimality Theory. While the model as described in the previous chapters does not match the explanatory power of non-linguistically based models such as De Boer (2001) or Schwartz et al. (1997b), I believe the results demonstrate two things. First, it shows that optimally dispersed vowel systems may be derived in a model based on stochastic grammars without assuming teleological 'dispersion' constraints, innate markedness constraints, or explicit storage of previously heard vowels, as has already been argued for the case of sibilants in Boersma & Hamann (2008) and for the specific case of five vowels in Boersma (2007). Second, it shows that OT-based models of learning can be integrated into agent-based, self-organizing artificial language simulations.

The model's predictions for vowel systems of sizes three to six are consistent with typical systems only with respect to some of the vowels. I do not believe this to be an intrinsic problem of the model, but rather an artifact of the parameters used in the simulation, in particular the calculations of effort values from the articulatory model. As is, the parameters of height, position and rounding are given equal weight in the calculation and are independent of one another. These assumptions might be overly simplistic to the point that they hurt the realism of the predicted vowel systems, and may have to be changed. The currently exaggerated preference of the model for the central close vowels, for instance, might be alleviated by changing the weightings of the rounding, height and position parameters. Likewise, the model's prediction that distinctions in formant values alone cannot be efficiently dispersed for more than six vowels is most likely a consequence of the noise values chosen in the simulation.

The predictions made by the models on vowel splits and mergers are also accurate in some respects and less accurate in others. Simulating a vowel merger resulted in a bona fide chain shift of all vowels due to the newly created 'gap' in the sound system. For the simulated split, results were less convincing. As with the vowel dispersion for more crowded vowel systems, I hypothesize that changing the noise levels might result in a more realistic outcome.

6.2 Limits of the current model

The previous section argues that some flaws in the simulation algorithm may be fixed by systematic exploration of different parameter settings. It would be interesting to see how far this exploration would take the model in terms of correctly predicting vowel inventories. However, as De Boer (2004) points out, the computer simulations are a *means* of doing research into hypotheses about language, not an end in itself. Endlessly tinkering with the parameters of the model until it produces the right result is research only into a model, not into language. With this paper, I hope to have shown that techniques and insights developed outside the field of linguistics can be integrated into linguistic models, in this case the model of Bidirectional Phonology and Phonetics.

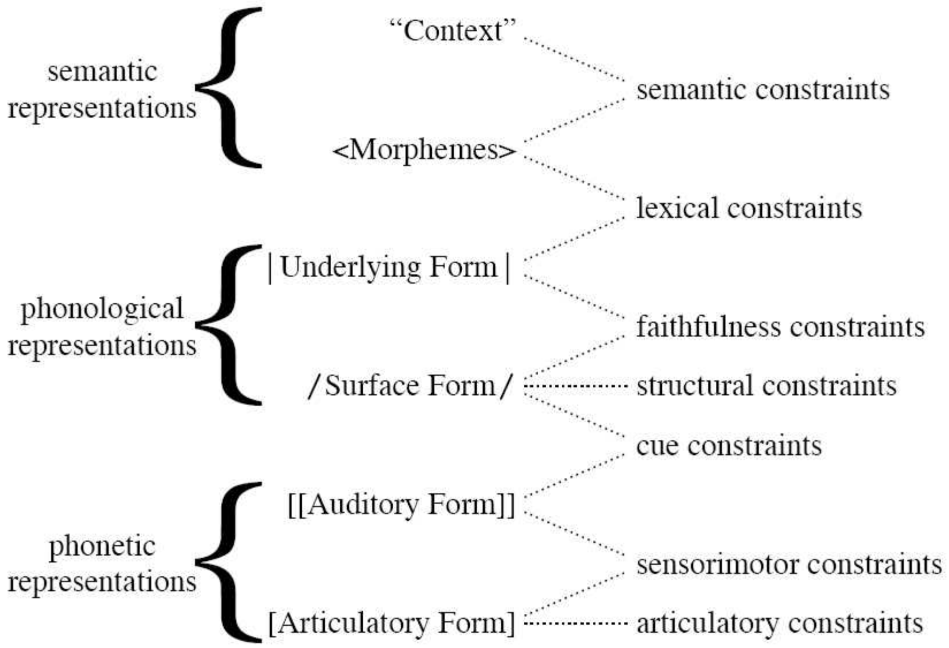


Figure 26: An overview of the representational levels in the Bidirectional Phonology and Phonetics *model, taken from Boersma (2008).*

I believe this integration can also work the other way around. The simulation described in the previous chapters operates only on the surface level of the phoneme and an underlying articulatory and acoustic representation, but that will only take us so far. Representing real language requires more levels of representation (see Figure 26); actual spoken language is obviously a lot more complex than single, time-invariant vowels. Full-scale phonological models like Bidirectional Phonology and Phonetics provide a formal means of explaining this complexity, ultimately explaining large-scale linguistic patterns and typology through small-scale processes at the level of the speaker, as has been done for the diachronic process of chain shifts here. It will probably be a worthwile endeavour to see if the ability of the model to explain diachronic and synchronic vowel processes can be improved by adding more levels of representation and a more sophisticated model of constraint and category development. Likewise, the linguistic tension between clarity and efficiency and the self-organization emerging from these opposing forces is not limited to vowels and could explored within bidirectional OT for other levels as well.

Acknowledgements

I thank Paul Boersma and Bart de Boer for their immediate willingness to act as my supervisor and “tweede lezer” respectively, and for their inspiring classes and personal encouragement which directly stimulated me to try my hand at simulating language evolution. My thanks also go out to Titia Benders and Paola Escudero, who have given me a venue for presenting my research to my fellow students.

To my parents I owe much gratitude for their support. For bearing with me during the writing of this thesis and providing the necessary distractions and perspective I thank my siblings, my friends, my bands St. Vincent & the Grenadines and Smook, and most importantly Arda.

References

Apoussidou, Diana (2007). *The learnability of metrical phonology*. PhD dissertation, University of Amsterdam.

Archangeli, Diana & D. Terence Langendoen [eds.] (1997). *Optimality Theory: An Overview.* Oxford: Blackwell.

Bladon, Anthony (1983). Two-formant models of vowel perception: Shortcomings and enhancements. *Speech Communication* 2. 305-313.

Blevins, Juliette (2004). Evolutionary Phonology: the emergence of sound patterns. Cambridge: Cambridge University Press.

Boersma, Paul (1997). How we learn variation, optionality, and probability. *Proceedings of the Institute of Phonetic Sciences of the University of Amsterdam* 21. 43–58.

Boersma, Paul (1998). *Functional phonology: formalizing the interactions between articulatory and perceptual drives*. PhD dissertation, University of Amsterdam.

Boersma, Paul (2007). *The emergence of auditory contrast*. Presentation at GLOW 30, Workshop on Segment Inventories, Tromsø, April 11 2007

Boersma, Paul (2008). *Emergent ranking of faithfulness explains markedness and licensing by cue*. ROA-684, Rutgers Optimality Archive, http://roa.rutgers.edu/

Boersma, Paul & Paola Escudero (2008). Learning to perceive a smaller L2 vowel inventory: An Optimality Theory account. In P. Avery, E Dresher & K. Rice [eds.]: *Contrast in phonology: Theory, Perception, Acquisition*. Mouton de Gruyter, Berlin.

Boersma, Paul & Silke Hamann (2008). The evolution of auditory dispersion in bidirectional constraint grammars. *Phonology* 25. 217-270.

Boersma, Paul, & Bruce Hayes (2001). Empirical tests of the Gradual Learning Algorithm. *Linguistic Inquiry* 32. 45–86.

Boersma, Paul & David Weenink (2008). Praat: doing phonetics by computer (Version 5.0.43) [Computer program]. Retrieved December 9, 2008, from http://www.praat.org/

Boersma, Paul, Paola Escudero & Rachel Hayes (2003). Learning abstract phonological from auditory phonetic categories: An integrated model for the acquisition of language-specific sound categories. *Proceedings of the 15th International Congress of Phonetic Sciences, Barcelona, August 3-9, 2003*. 1013-1016

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

de Boer, Bart (2001). *The Origins of Vowel Systems*. Oxford Linguistics. Oxford University Press.

de Boer, Bart (2006). Computer modeling as a tool for understanding language evolution. In N. Gonthier, J.P. van Bendegem & D. Aerts [eds.]:*Evolutionary Epistemology, Language and Culture – A nonadaptationist systems theoretical approach.* Dordrecht: Springer.

Dediu, Dan & D. Robert Ladd (2007). Linguistic tone is related to the population frequency of the adaptive haplogroups of two brain size genes, ASPM and Microcephalin. *Proceedings of the National Academy of Sciences of the USA* 104. 10944-10949.

Escudero, Paola & Paul Boersma (2004). Bridging the gap between L2 speech perception research and phonological theory. *Studies in Second Language Acquisition* 26. 551–585

Everett, Daniel (2005). Cultural constraints on grammar and cognition in Pirahã. *Current Anthropology* 46. 621–47.

Flemming, Edward (2002). *Auditory Representations in Phonology***.** New York: Routledge.

Kager, René (1999). *Optimality Theory.* Cambridge: Cambridge University Press.

Kirchner, Robert (1998). *Some new generalizations concerning geminate inalterability.* Handout at a meeting of the Linguistic Society of America. ROA-240, Rutgers Optimality Archive, http://roa.rutgers.edu/

Liljencrants, Johan & Björn Lindblom (1972). Numerical simulation of vowel quality systems: the role of perceptual contrast. *Language* 48. 839–862 (1972).

Maddieson, Ian (1984). *Patterns of Sounds*. Cambridge: Cambridge University Press.

Maeda, Shinji (1989). Compensatory articulation during speech: evidence from the analysis and synthesis of vocal tract shapes using an articulatory model. In W. J. Hardcastle & A. Marchal, [eds.]: *Speech production and speech modelling*. 131-149. Dordrecht: Kluwer.

Ohala, John J. (1981). The listener as a source of sound change. In C. S. Masek, R. A. Hendrick & M. F. Miller [eds.]: *Papers from the parasession on language and behavior.* Chicago: Chicago Linguistic Society. 178–203.

Prince, Alan & Paul Smolensky (1993). *Optimality Theory: constraint interaction in generative grammar.* Manuscript, Rutgers University & University of Colorado, Boulder. Published 2004, Malden, Massachusetts & Oxford: Blackwell.

Schwartz, Jean-Luc, Louis-Jean Boë, Nathalie Vallée & Christian Abry (1997a). Major trends in vowel system inventories, *Journal of Phonetics* 25. 233-253.

Schwartz, Jean-Luc, Louis-Jean Boë, Nathalie Vallée & Christian Abry (1997b). The dispersion-focalization theory of vowel systems. *Journal of Phonetics* 25. 255-286.

Steriade, Donca (2001). Directional asymmetries in place assimilation. In Elizabeth Hume & Keith Johnson [eds.]: *The role of speech perception in phonology.* San Diego: Academic Press. 219-250.

Stevens, Kenneth (1972).The quantal nature of speech: evidence from articulatory-acoustic data. In *Human communication: a unified view* (E. E. David, Jr & P. B. Denes, [eds.]). 51-66. New York: McGraw-Hill.

Ten Bosch, Louis (1991). *On the structure of vowel systems. Aspects of an extended vowel model using effort and contrast*. Doctoral dissertation, University of Amsterdam.

Traill, Anthony (1985). *Phonetic and phonological studies of !Xóõ Bushman*. Hamburg: Helmut Buske.

Vallée, Nathalie (1994). *Systemes vocaliques: de la typologie aux predictions*. PhD dissertation, Université Stendhal, Grenoble

van Leussen, Jan-Willem & Maaike Vondenhoff (2008). *A self-organizational approach to the Gradual Learning Algorithm in modelling the evolution of auditory contrast.* Unpublished *s*tudent paper for course 'Evolution of Speech', University of Amsterdam. [Available from the author of this thesis.]

Wedel, Andrew (2006). Exemplar models, evolution and language change. *The Linguistic Review* 23. 247–274.

Appendix I – Java code used in the simulation

Evo2dim.java

```
/*
 * This is the main class that is executed to run a simulation.
 * It passes the values of the parameters to an instance of the World class,
 * which contains a simulation world.
 */

import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.text.SimpleDateFormat;
import java.util.Calendar;

public class Evo2dim {
	public static double impossible = 105.0;
	public static double easiestRank = 90.0;
	public static double hardestRank = 105.0;
	public static double evalNoise = 2.0;
	public static double transNoiseF1 = 0.25;
	public static double transNoiseF2 = 0.75;
	public static double plasticity = 0.02;
	public static int epochTime = 10000;
	public static int numberOfEpochs = 20;
	public static boolean initialize = true;
	public static int numberOfPhonemes = 5;
	public static String outputDir;
	public static String grammarOutputDir;
	public static void main(final String[] args) throws IOException {
		Calendar cal = Calendar.getInstance();
		SimpleDateFormat sdf = new SimpleDateFormat("yyMMdd_HHmm");
		String dirDate = sdf.format(cal.getTime());
		outputDir = dirDate +"\\";
		grammarOutputDir = outputDir+"grammars\\";

		// Attempt to create the output directories
		boolean success = new File(outputDir).mkdir();
		if (!success) {
			System.out.println("Could not create output directory!");
			System.exit(0);
		}
		success = new File(outputDir +"pics\\").mkdir();
		if (!success) {
			System.out.println("Could not create dir \"pics\"!");
			System.exit(0);
		}
		success = new File(outputDir+"sounds\\").mkdir();
		if (!success) {
			System.out.println("Could not create dir \"sounds\"!");
			System.exit(0);
		}
		success = new File(outputDir+"grammars\\").mkdir();
		if (!success) {
			System.out.println("Could not create dir \"grammars\"");
			System.exit(0);
		}
		File effortTable = new File(outputDir+"effortRankings.txt");
		PrintWriter effort = new PrintWriter(effortTable);
		World world = new World(transNoiseF1, transNoiseF2, epochTime, numberOfEpochs);
		double[][] rankings = Grammar.calcArtConstraints();

		// Print the articulatory constraints to a text file
		for (int i=0; i < World.stepsF1; i++)
		{
```

```java
                        for (int j=0; j < World.stepsF2; j++)
                        {
                              effort.print(rankings[(World.stepsF1-1)-i][(World.stepsF2-1)-j]);
                              if (j<(World.stepsF2-1)) {
                                    effort.print(" ");
                              }
                        }
                        effort.println();
                  }
                  effort.close();
                  world.runSimulation(numberOfPhonemes, initialize);
            }
      }
```

World.java

```java
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Random;

public class World {
	File outputFile;
	File grammarOutputFile;
	BufferedWriter out;
	Random random;
	static Timer time;
	static int numPhonemes;
	double transNoiseF1;
	double transNoiseF2;
	int repsPerEpoch;
	int numEpochs;
	public static int[] phonemes;
	public static int maxSize = 10;
	public static double minF1 = 2.25;
	public static double maxF1 = 7.25;
	public static double minF2 = 5.5;
	public static double maxF2 = 15.5;
	public static int stepsF1 = 21;
	public static int stepsF2 = 21;
	public static double stepsizeF1 = (double) (maxF1 - minF1) / (stepsF1 - 1);
	public static double stepsizeF2 = (double) (maxF2 - minF2) / (stepsF2 - 1);
	public static Constraint[] artConstraints = Grammar.createArtConstraints();
	public static Candidate[][] productionCandidates;
	public static ArrayList<Agent> agents;

	public World(double transNoiseF1_, double transNoiseF2_, int repsPerEpoch_,
			int numEpochs_) {
		random = new Random();
		this.transNoiseF1 = transNoiseF1_;
		this.transNoiseF2 = transNoiseF2_;
		this.repsPerEpoch = repsPerEpoch_;
		this.numEpochs = numEpochs_;
		time = new Timer();

	}

	public void runSimulation(int numPhon, boolean initial) throws IOException {

		numPhonemes = numPhon;
		int agentNumber = 0;

		System.out.println("** STARTING SIMULATION WITH " + numPhonemes
				+ " PHONEMES **");

		final double[] means_f1 = { 3.8, 4.12, 5.6, 3.2 };
		final double[] means_f2 = { 7.5, 13.5, 11.75, 11.5 };

		int speakingAge = 1 * repsPerEpoch;
		int dyingAge = 2 * repsPerEpoch;

		phonemes = new int[numPhonemes];

		for (int f = 0; f < numPhonemes; f++) {
			phonemes[f] = f;
		}
		productionCandidates = new Candidate[numPhonemes][stepsF1 * stepsF2];
```

```
        for (int phonNum = 0; phonNum < numPhonemes; phonNum++) {
            int canNum = 0;
            for (int stepF1 = 0; stepF1 < World.stepsF1; stepF1++) {
                for (int stepF2 = 0; stepF2 < World.stepsF2; stepF2++) {
                    double[] canValues = new double[2];
                    canValues[0] = minF1 + (stepsizeF1 * stepF1);
                    canValues[1] = minF2 + (stepsizeF2 * stepF2);
                    Candidate can = new Candidate(phonNum, canValues);
                    productionCandidates[phonNum][canNum] = can;
                    canNum++;
                }
            }
        }

        agents = new ArrayList<Agent>(10);

        // Create a log file storing the settings
        java.io.File logFile = new java.io.File(Evo2dim.outputDir + numPhonemes
                + "phonemes.txt");
        if (logFile.exists()) {
            System.out.println("Output file already exists!");
            System.exit(0);
        }
        java.io.PrintWriter logText = new PrintWriter(logFile);
        logText.println("** Properties of the created table files **");
        logText.println("Number of phonemes: " + numPhonemes);
        logText.println("Horizontal transmission noise: " + transNoiseF1);
        logText.println("Vertical transmission noise: " + transNoiseF2);
        logText.println("Evaluation noise: " + Evo2dim.evalNoise);
        logText.println("Plasticity: " + Evo2dim.plasticity);
        logText.println("Minimum/maximum F1 value and step size: [" + minF1 + "-"
                + maxF1 + "], " + stepsizeF1);
        logText.println("Minimum/maximum F2' value and step size: [" + minF2 + "-"
                + maxF2 + "], " + stepsizeF2);
        logText.println("Minimal ranking for possible articulation: "
                + Evo2dim.easiestRank);
        logText.println("Minimal ranking for possible articulation: "
                + Evo2dim.hardestRank);
        logText.println("Ranking for impossible articulation: "
                + Evo2dim.impossible);
        logText.println("Number of timesteps in simulation: " + numEpochs
                * repsPerEpoch);
        logText.println("Agents start speaking at age " + speakingAge);
        logText.println("Agents die at age " + dyingAge);
        if (initial) {
            logText.println("Initial values given:");
            for (int i = 0; i < means_f1.length; i++) {
                logText.print(" " + means_f1[i]);
                logText.println("/" + means_f2[i]);
            }
        }
        logText.close();
        /*
         * Main loop
         */
        for (int timestep = 0; timestep < numEpochs * repsPerEpoch; timestep++) {

            if ((timestep % (int) (repsPerEpoch / 10) == 0)) {
                System.out.println("Now at timestep " + timestep);
                for (Agent a : agents) {
                    grammarOutputFile = new
java.io.File(Evo2dim.grammarOutputDir
                            + timestep + "_" + "_" + a + "_" + numPhonemes
+ "phonemes.tbl");
                    a.grammar.printGrammar(grammarOutputFile);
```

```
                                        }
                                        if (timestep % (repsPerEpoch / 5) == 0) {
                                                if (timestep > speakingAge)
                                                        out.close();
                                                if (timestep >= speakingAge) {
                                                        outputFile = new java.io.File(Evo2dim.outputDir +
timestep + "_"
                                                                        + numPhonemes + "phonemes.tbl");
                                                        out = new BufferedWriter(new
FileWriter(outputFile));
                                                        out.write("time,agent,age,phoneme,F1,F2");
                                                        out.newLine();
                                                }
                                        }
                                }

                                // Speaking stage
                                for (Agent currAgent : agents) {
                                        if (currAgent != null && currAgent.age >= speakingAge) {
                                                for (int numTokens = 0; numTokens < numPhonemes; numTokens+
+) {
                                                        Candidate speakWinner = currAgent
                                                                        .speak(random.nextInt(numPhonemes));
                                                        double[] canValues = {
                                                                        speakWinner.values[0] +
random.nextGaussian() * transNoiseF1,
                                                                        speakWinner.values[1] +
random.nextGaussian() * transNoiseF2 };
                                                        out.write(timestep + "," + currAgent + "," +
currAgent.age + ","
                                                                        + phonName(speakWinner.phon) + "," +
canValues[0] + ","
                                                                        + canValues[1]);
                                                        Candidate toAdd = new Candidate(speakWinner.phon,
canValues);
                                                        addToInput(currAgent, toAdd, agents);
                                                        out.newLine();
                                                }
                                        }
                                }

                                if (initial) {
                                        double value_hor = 0;
                                        double value_ver = 0;
                                        for (int phonNum = 0; phonNum < phonemes.length; phonNum++) {
                                                int randPhon = random.nextInt(numPhonemes);
                                                value_hor = means_f1[randPhon] + random.nextGaussian() *
transNoiseF1;
                                                value_ver = means_f2[randPhon] + random.nextGaussian() *
transNoiseF2;
                                                double[] canValues = { value_hor, value_ver };
                                                Candidate toAdd = new Candidate(randPhon, canValues);
                                                addToInput(null, toAdd, agents);
                                        }
                                }

                                if (timestep == 1 * repsPerEpoch) {
                                        initial = false;
                                        System.out.println("Initial training over.");
                                }

                                // Listening and aging stage
                                for (int agentCount = 0; agentCount < agents.size(); agentCount++) {
                                        // Listening
                                        Agent currAgent = agents.get(agentCount);
                                        currAgent.listen();
```

```
					// Aging
					currAgent.age += 1;

					if (currAgent.age == dyingAge) {
						System.out.println(currAgent + " dies.");
						agents.remove(agentCount);

					}
				}

				if (timestep % (repsPerEpoch) == 0) {
					if (agents.size() <= maxSize) {
						addNewAgent(agentNumber);
						agentNumber++;
					}

				}
			}
			out.close();
		}

		private static void addToInput(Agent speakingAgent, Candidate toAdd,
				ArrayList<Agent> agents) {
			for (int counter = 0; counter < agents.size(); counter++) {
				if (agents.get(counter) != speakingAgent) {
					agents.get(counter).inputs.add(toAdd);
				}
			}
		}

		public void addNewAgent(int number) {
			Agent a = new Agent(number, phonemes);
			agents.add(a);
			agents.trimToSize();
		}

		public static void printArray(int[] array) {
			for (int i = 0; i < array.length; i++) {
				System.out.print(array[i] + ", ");
			}
		}

		public static char phonName(int i) {
			return (char) (65 + i);
		}

		public static double[] roundToNearest(double[] values) {
			if (values[0] < minF1) {
				values[0] = minF1;
			} else if (values[0] > maxF1) {
				values[0] = maxF1;
			} else {
				values[0] = stepsizeF1 * ((int) (0.5 + (values[0] / stepsizeF1)));
			}
			if (values[1] < minF2) {
				values[1] = minF2;
			} else if (values[0] > maxF2) {
				values[1] = maxF2;
			} else {
				values[1] = stepsizeF2 * ((int) (0.5 + (values[1] / stepsizeF2)));
			}
			return values;
		}
	}
```

Agent.java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

public class Agent {
      Grammar grammar;
      int age;
      String name;
      int[] phonemes;
      ArrayList<Candidate> inputs;

      /*
       * Constructor method. Creates an agent with a 'blank slate' grammar.
       */
      Agent(int number, int[] phon) {
            phonemes = phon;
            grammar = new Grammar(phonemes);
            age = 0;
            name = ("Agent" + number);
            inputs = new ArrayList<Candidate>();
            System.out.println(name + " is born.");
      }

      /*
       * @param target Processes (listens to) a candidate and check if
       * a learning step must be undertaken.
       */
      public void process(Candidate target) {
            // noisyGrammar.printGrammar();
            ArrayList<Candidate> candidates = new ArrayList<Candidate>(World.agents
                        .size());
            for (int phonNum = 0; phonNum < phonemes.length; phonNum++) {
                  Candidate can = new Candidate(phonemes[phonNum], target.values);
                  candidates.add(can);
            }
            Candidate winner = grammar.evaluate(candidates, false);
            if (winner.phon != target.phon) {
                  //System.out.println(this + " heard " + winner.phon + " instead of "
                  //            + target.phon);
                  grammar.updateGrammar(target, winner, this.age);
            }
      }

      /*
   * Not used in current implementation
       */
      public double[][] testProduction(int phon, int size) {
            double[][] outputs = new double[2][size];
            System.out.print("Testing " + this.toString() + ",phoneme " + phon);
            Candidate winner;
            for (int g = 0; g < size; g++) {
                  winner = this.speak(phon);
                  outputs[0][g] = winner.values[0];
                  outputs[1][g] = winner.values[1];
                  if (g % (size / 10) == 0) {
                        System.out.print(".");
                  }
            }
            System.out.println("done.");
            return outputs;
      }

      /*
```

```
	 * Speaks a phoneme throguh the Grammar.evaluate method
	 */
	public Candidate speak(int phonNum) {
		List<Candidate> list = Arrays.asList(World.productionCandidates[phonNum]);
		ArrayList<Candidate> candidates = new ArrayList<Candidate>(list);
		// noisyGrammar.printGrammar();
		Candidate winner = grammar.evaluate(candidates, true);
		// System.out.println(this + " says: " + winner);
		return winner;

	}

	public void listen() {
		for (int i = 0; i < inputs.size(); i++) {
			this.process(inputs.get(i));
			// System.out.println(this.toString() + "hears token number" + i);
		}
		inputs.clear();
	}

	public String toString() {
		return name;
	}
}
```

Grammar.java

```
import java.io.File;
import java.io.FileNotFoundException;
import java.io.PrintWriter;
import java.util.ArrayList;
import java.util.Arrays;

public class Grammar {

	public double plasticity = Evo2dim.plasticity;
	int numCueConstraints = (int) World.numPhonemes
			* (World.stepsF1 + World.stepsF2);
	int numArtConstraints = (int) World.stepsF1 * World.stepsF2;
	Constraint[] cueConstraints;
	Constraint[] allConstraints;

	/*
	 * Constructor method
	 */
	Grammar(int[] phonemes) {
		System.out.println("Number of art constraints:" + numArtConstraints);
		System.out.println("Number of cue constraints:" + numCueConstraints);
		double startRank = 100;
		double[] values;
		int arrayKey = 0;
		cueConstraints = new Constraint[numCueConstraints];
		allConstraints = new Constraint[numCueConstraints + numArtConstraints];
		for (int phonNum = 0; phonNum < phonemes.length; phonNum++) {

			// Create F1 cue constraints
			for (int f = 0; f < World.stepsF1; f++) {
				values = new double[2];
				values[0] = World.minF1 + (World.stepsizeF1 * f);
				cueConstraints[arrayKey] = new Constraint(true, phonNum, 0,
values,
						startRank);
				arrayKey++;
			}

			// Create F2 cue constraints
			for (int f = 0; f < World.stepsF2; f++) {
				values = new double[2];
				values[1] = World.minF2 + (World.stepsizeF2 * f);
				cueConstraints[arrayKey] = new Constraint(true, phonNum, 1,
values,
						startRank);
				arrayKey++;
			}
		}
		System.arraycopy(cueConstraints, 0, allConstraints, 0, numCueConstraints);
		System.arraycopy(World.artConstraints, 0, allConstraints,
				numCueConstraints, numArtConstraints);
		System.out.println("Copying constraints...");
	}

	public void noisify(Constraint[] cons) {
		for (Constraint c : cons) {
			c.noisify();
		}
	}

	/*
	 * An alternative method of calculating articulatory constraints, based on
	 * Boersma (2007), not used in the current code.
	 */
```

```java
public double oldCalcArtRank(int[] values) {
	double x = values[0];
	double y = values[1];
	double a = Math.pow((1 - x / 50), 2);
	double b = Math.pow((1 - y / 200), 2);
	double c = Math.pow((a + b), 3);
	double d = Math.pow((16 * c), 3);
	double e = Math.pow((y / 97), 6);
	double f = Math.pow((16 * e), 3);
	double rank = 87 + Math.pow((d + f), (0.3333333));
	return rank;
}

/*
 * Prints a grammar
 */
public void printGrammar(File output) throws FileNotFoundException {
	PrintWriter out = new PrintWriter(output);
	out.println("Size: " + allConstraints.length);
	for (Constraint c : cueConstraints) {
		out.println(c + " : " + c.ranking);
	}
	out.close();
}

/*
 * Updates the grammar by demoting and promoting constraints responsible for
 * candidate 'winner' winning instead 'target' winning.
 */
public void updateGrammar(Candidate target, Candidate winner, int age) {
	int dim;
	double shift = plasticity;
	for (Constraint c : cueConstraints) {
		dim = c.dimension;
		if (c.values[dim] == target.values[dim]) {
			if (c.phoneme == target.phon) {
				c.ranking -= shift;
				// System.out.println(c + " is demoted");
			}
			if (c.phoneme == winner.phon) {
				c.ranking += shift;
				// System.out.println(c + " is promoted");
			}
		}
	}
}

/*
 * Evaluates an ArrayList of candidates. The boolean parameter speak
 * determines whether it is a production (speak=true) or a perception
 * (speak=false) evaluation.
 */
public Candidate evaluate(ArrayList<Candidate> candidates, boolean speak) {
	Constraint[] evalGrammar;
	if (speak)
		evalGrammar = allConstraints;
	else
		evalGrammar = cueConstraints;
	noisify(evalGrammar);
	Arrays.sort(evalGrammar);
	Constraint currCon;
	Candidate can;
	ArrayList<Integer> violators = new ArrayList<Integer>((World.stepsF1));
	for (int f = 0; candidates.size() > 1; f++) {
		currCon = (evalGrammar[f]);
		int numCans = candidates.size();
```

```
                    for (int currCan = 0; currCan < numCans; currCan++) {
                            can = candidates.get(currCan);
                            if (currCon.violatedBy(can)) {
                                    violators.add(currCan);
                            }
                    }
                    int numViolations = violators.size();
                    if (numViolations > 0) {
                            if (numCans > numViolations) {
                                    for (int removeIndex = numViolations - 1; removeIndex >= 0;
removeIndex--) {
                                            int toRemove = violators.get(removeIndex);
                                            candidates.remove(toRemove);
                                    }
                            }
                            violators.clear();
                    }
            }
            Candidate winner = candidates.get(0);
            return winner;
    }
  /* A method that used Harmonic Grammar (weighted constraints) instead of
     * Optimality Theory to evaluate candidates. Not implemented in current code.
     */
    public Candidate evaluateHG(ArrayList<Candidate> candidates, boolean speak) {
            Constraint[] evalGrammar;
            if (speak)
                    evalGrammar = allConstraints;
            else
                    evalGrammar = cueConstraints;
            noisify(evalGrammar);
            Arrays.sort(evalGrammar);
            Candidate can;
            int numCans = candidates.size();
            double[] scores = new double[numCans];
            for (int i = 0; i < numCans; i++) {
                    scores[i] = 0;
            }
            for (Constraint c : evalGrammar) {
                    for (int currCan = 0; currCan < numCans; currCan++) {
                            can = candidates.get(currCan);
                            if (c.violatedBy(can)) {
                                    System.out.print("Candidate " + can);
                                    System.out.println(" violates constraint " + c);
                                    scores[currCan] -= c.noisyRanking;
                            }
                    }
            }
            double lowestValue = 0;
            int winnerNumber = 0;
            for (int i = 0; i < numCans; i++) {
                    if (scores[i] < lowestValue) {
                            lowestValue = scores[i];
                            winnerNumber = i;
                    }

            }
            Candidate winner = candidates.get(winnerNumber);
            return winner;
    }

    /*
     * Adds articulatory constraints to a grammar
     */
    static Constraint[] createArtConstraints() {
            int numArtConstraints = (int) World.stepsF1 * World.stepsF2;
		double[][] rankings = calcArtConstraints();
		Constraint[] constraints = new Constraint[numArtConstraints];
		int arrayKey = 0;
		for (int i = 0; i < World.stepsF1; i++) {
			for (int j = 0; j < World.stepsF2; j++) {
				double[] values = new double[2];
				values[0] = World.minF1 + (World.stepsizeF1 * i);
				values[1] = World.minF2 + (World.stepsizeF2 * j);
				double artRank = rankings[i][j];
				constraints[arrayKey] = new Constraint(false, -1, 0, values,
artRank);
				arrayKey++;
			}
		}
		return constraints;
	}
/*
 * An alternative method that uses average effort for an F1,F2 value pair
 * instead of least effort. Not used in this version of the code.
 */
	static double[][] calcArtConstraints_avg() {

		double[][] rankings = new double[World.stepsF1][World.stepsF2];
		int[][] counts = new int[World.stepsF1][World.stepsF2];
		double impossible = Evo2dim.impossible;
		double easiest = Evo2dim.easiestRank;
		double hardest = Evo2dim.hardestRank;
		double diffConstant = hardest - easiest;

		for (int i = 0; i < World.stepsF1; i++) {
			for (int j = 0; j < World.stepsF2; j++) {
				rankings[i][j] = 0;
				counts[i][j] = 0;
			}
		}

		int drawingGranularity = 40;
		for (int h_i = 0; h_i <= drawingGranularity; h_i++) {
			for (int p_i = 0; p_i <= drawingGranularity; p_i++) {
				for (int r_i = 0; r_i <= drawingGranularity; r_i++) {
					double h = h_i / (double) drawingGranularity;
					double p = p_i / (double) drawingGranularity;
					double r = r_i / (double) drawingGranularity;
					double h_effort = Math.abs(0.5 - h);
					double p_effort = Math.abs(0.5 - p);
					double r_effort = Math.abs(0.5 - r);
					double avgEffort = (h_effort + p_effort + r_effort) / 1.5;

					double f1 = ((-392 + 392 * r) * Math.pow(h, 2) + (596 - 668
* r) * h + (-146 + 166 * r))
							* Math.pow(p, 2)
							+ ((348 - 348 * r) * Math.pow(h, 2) + (-494 +
606 * r) * h + (141 - 175 * r))
							* p
							+ ((340 - 72 * r) * Math.pow(h, 2) + (-796 +
108 * r) * h + (708 - 38 * r));

					double f2 = ((-1200 + 1208 * r) * Math.pow(h, 2) + (1320 -
1328 * r)
							* h + (118 - 158 * r))
							* Math.pow(p, 2)
							+ ((1864 - 1488 * r) * Math.pow(h, 2) + (-2644
+ 1510 * r) * h + (-561 + 221 * r))
							* p
							+ ((-670 + 490 * r) * Math.pow(h, 2) + (1355 -
697 * r) * h + (1517 - 117 * r));
```

```
                                double f3 = ((604 - 604 * r) * Math.pow(h, 2) + (1038 -
1178 * r) * h + (246 + 566 * r))
                                             * Math.pow(p, 2)
                                             + ((-1150 + 1262 * r) * Math.pow(h, 2) + (-
1443 + 1313 * r) * h + (-317 - 483 * r))
                                             * p
                                             + ((1130 - 836 * r) * Math.pow(h, 2) + (-315 +
44 * r) * h + (2427 - 127 * r));

                                double f4 = ((-1120 + 16 * r) * Math.pow(h, 2) + (1696 -
180 * r) * h + (500 + 522 * r))
                                             * Math.pow(p, 2)
                                             + ((-140 + 240 * r) * Math.pow(h, 2) + (-578 +
214 * r) * h + (-692 - 419 * r))
                                             * p
                                             + ((1480 - 602 * r) * Math.pow(h, 2) + (-1220
+ 289 * r) * h + (3678 - 178 * r));

                                f1 = Functions.hertzToBark(f1);
                                f2 = Functions.hertzToBark(f2);
                                f3 = Functions.hertzToBark(f3);
                                f4 = Functions.hertzToBark(f4);
                                double f2prime = Functions.effectiveF2(f2, f3, f4);
                                double[] f1f2 = { f1, f2prime };
                                f1f2 = World.roundToNearest(f1f2);
                                int f1cell = (int) ((f1f2[0] - World.minF1) /
World.stepsizeF1);
                                int f2cell = (int) ((f1f2[1] - World.minF2) /
World.stepsizeF2);
                                rankings[f1cell][f2cell] += avgEffort;
                                counts[f1cell][f2cell]++;
                          }
                    }
             }
             for (int i = 0; i < World.stepsF1; i++) {
                    for (int j = 0; j < World.stepsF2; j++) {
                          int currCount = counts[i][j];
                          double currRank = rankings[i][j];
                          if (currCount > 0)
                                rankings[i][j] = easiest + (diffConstant * (currRank /
currCount));
                          else
                                rankings[i][j] = impossible;
                    }
             }
             return rankings;
      }
  /*
   * The method used to calculate the articulatory constraints, based on the
   * articulatory synthesis used in De Boer (2001)
   */
      static double[][] calcArtConstraints() {

             double[][] rankings = new double[World.stepsF1][World.stepsF2];
             double impossible = Evo2dim.impossible;
             double easiest = Evo2dim.easiestRank;
             double hardest = Evo2dim.hardestRank;
             double diffConstant = hardest - easiest;

             for (int i = 0; i < World.stepsF1; i++) {
                    for (int j = 0; j < World.stepsF2; j++) {
                          rankings[i][j] = impossible;
                    }
             }
             int drawingGranularity = 40;
             for (int h_i = 0; h_i <= drawingGranularity; h_i++) {
                   for (int p_i = 0; p_i <= drawingGranularity; p_i++) {
                          for (int r_i = 0; r_i <= drawingGranularity; r_i++) {
                                double h = h_i / (double) drawingGranularity;
                                double p = p_i / (double) drawingGranularity;
                                double r = r_i / (double) drawingGranularity;
                                double h_effort = Math.abs(0.5 - h);
                                double p_effort = Math.abs(0.5 - p);
                                // double r_effort = Math.abs(0.5-r);
                                double r_effort = (r / 2);
                                double avgEffort = (h_effort + p_effort + r_effort) / 1.5;

                                double f1 = ((-392 + 392 * r) * Math.pow(h, 2) + (596 - 668
* r) * h + (-146 + 166 * r))
                                             * Math.pow(p, 2)
                                             + ((348 - 348 * r) * Math.pow(h, 2) + (-494 +
606 * r) * h + (141 - 175 * r))
                                             * p
                                             + ((340 - 72 * r) * Math.pow(h, 2) + (-796 +
108 * r) * h + (708 - 38 * r));

                                double f2 = ((-1200 + 1208 * r) * Math.pow(h, 2) + (1320 -
1328 * r)
                                             * h + (118 - 158 * r))
                                             * Math.pow(p, 2)
                                             + ((1864 - 1488 * r) * Math.pow(h, 2) + (-2644
+ 1510 * r) * h + (-561 + 221 * r))
                                             * p
                                             + ((-670 + 490 * r) * Math.pow(h, 2) + (1355 -
697 * r) * h + (1517 - 117 * r));

                                double f3 = ((604 - 604 * r) * Math.pow(h, 2) + (1038 -
1178 * r) * h + (246 + 566 * r))
                                             * Math.pow(p, 2)
                                             + ((-1150 + 1262 * r) * Math.pow(h, 2) + (-
1443 + 1313 * r) * h + (-317 - 483 * r))
                                             * p
                                             + ((1130 - 836 * r) * Math.pow(h, 2) + (-315 +
44 * r) * h + (2427 - 127 * r));

                                double f4 = ((-1120 + 16 * r) * Math.pow(h, 2) + (1696 -
180 * r) * h + (500 + 522 * r))
                                             * Math.pow(p, 2)
                                             + ((-140 + 240 * r) * Math.pow(h, 2) + (-578 +
214 * r) * h + (-692 - 419 * r))
                                             * p
                                             + ((1480 - 602 * r) * Math.pow(h, 2) + (-1220
+ 289 * r) * h + (3678 - 178 * r));

                                f1 = Functions.hertzToBark(f1);
                                f2 = Functions.hertzToBark(f2);
                                f3 = Functions.hertzToBark(f3);
                                f4 = Functions.hertzToBark(f4);
                                double f2prime = Functions.effectiveF2(f2, f3, f4);
                                double[] f1f2 = { f1, f2prime };
                                f1f2 = World.roundToNearest(f1f2);
                                int f1cell = (int) ((f1f2[0] - World.minF1) /
World.stepsizeF1);
                                int f2cell = (int) ((f1f2[1] - World.minF2) /
World.stepsizeF2);
                                double ranking = easiest + (diffConstant * avgEffort);
                                if (rankings[f1cell][f2cell] >= ranking) {
                                       rankings[f1cell][f2cell] = ranking;
                                }
                          }
```

```
                        }
                }
                return rankings;
        }
}
```

Constraint.java

```java
import java.util.Random;

/*
 / Constraints come in two types (isCue=false or true)
 */
public class Constraint implements Comparable<Constraint> {
	boolean isCue;
	int phoneme;
	int dimension;
	double[] values;
	double ranking;
	double noisyRanking;
	public static double evalNoise = Evo2dim.evalNoise;
	static Random r = new Random();

	Constraint(boolean _isCue, int _phoneme, int _dimension, double[] _values,
			double _ranking) {
		isCue = _isCue;
		phoneme = _phoneme;
		dimension = _dimension;
		values = _values;
		ranking = _ranking;
	}

	/*
	 * This method takes a candidate as input and checks if it is violated by that
	 * candidate.
	 */
	boolean violatedBy(Candidate candidate) {
		if (!isCue && values[0] == candidate.values[0]
								&& values[1] ==
candidate.values[1]) {
			return true;
		} else if (isCue && phoneme == candidate.phon
					&& values[dimension] == candidate.values[dimension]) {
			return true;
		}

		else {
			return false;
		}
	}

	// Adds Gaussian evaluation noise to a ranking
	public void noisify() {
		double noiseWeight = r.nextGaussian();
		double addedValue = (evalNoise * noiseWeight);
		noisyRanking = ranking + (addedValue);
	}

	public String toString() {
		String dimName;
		if (dimension == 0) {
			dimName = "F1";
		} else {
			dimName = "F2";
		}
		if (isCue) {
			return "[" + dimName + "=" + values[dimension] + "] */"
			+ World.phonName(phoneme) + "/";
		} else {
			return "*[F1=" + values[0] + " F2=" + values[1] + "]";
		}
	}
	/*
	 * Clones a Constraint object
	 */
	public Constraint copy() {
		double[] v = new double[2];
		v[0] = values[0];
		v[1] = values[1];
		return new Constraint(isCue, phoneme, dimension, v, ranking);
	}

	/*
	 * Necessary to use Java's internal sorting
	 */
	public int compareTo(Constraint c) {
		return (int) (c.noisyRanking - noisyRanking);
	}
}
```

Candidate.java

```java
/*
 * A Candidate object just has fields indicating phoneme type
 * and formant values.
 */
public class Candidate {
    int phon;
    double[] values = new double[2];

    /*
     * Constructor method
     */
    Candidate(int ph, double[] unroundedValues) {
       phon = ph;
      double[] roundedValues = World.roundToNearest(unroundedValues);
      values[0] = roundedValues[0];
      values[1] = roundedValues[1];
    }

    public String toString() {
        return "/" + phon + "/ [F1=" + values[0] + " F2=" + values[1] + "]";
    }
}
```

Functions.java

```java
/* This class contains a function for converting Hz values to Bark,
 * (formula copied from the Praat manual) and a function for calculating
 * the effective second formant from F2, F3 and F4 values (formula copied De Boer 2001)
 */
public class Functions {

      static public double hertzToBark(double hertz) {
            return 7*Math.log(hertz/650 + Math.sqrt(1 + Math.pow((hertz/650),2) ) );
      }

      static public double effectiveF2(double f2, double f3, double f4) {
            double c = 3.5;
            double w1 = (c-(f3-f2))/c;
            double w2 = ((f4-f3)-(f3-f2))/(f4-f2);
            if (f3 - f2 > c)
               return f2;
            else if (f3 - f2 <= c && f4 - f2 > c)
               return ((2-w1)*f2+w1*f3)/2;
            else if (f4 - f2 <= c && f3 - f2 < f4 - f3)
               return (( (w2*f2) + (2 - w2) * f3 ) / 2) - 1;
            else return (( (2+w2)*f3 - (w2*f4)) / 2) - 1;
      }
}
```