VRIJE
UNIVERSITEIT
BRUSSEL

SELF-ORGANIZATION IN VOWEL SYSTEMS FROM SMALL COMMUNITIES

Extending de Boer 2000 - EoS project

Lennert Bontinck

January, 2021-2022

Student number: 0568702
Computer Science: AI

Contents

1 Project goal and supplied code **1**

1.1 Project goal 1

1.2 Important files 2

2 Relevant literature **3**

2.1 Summary of de Boer (2000) 3

2.2 Importance of network structure for emergence 4

3 Re-implementing de Boer (2000) **5**

3.1 Producing sounds 5

3.2 Perceiving sounds 6

3.3 Representing agents 7

3.4 Playing and analysing games 9

4 Resolving some ad hoc decisions from de Boer (2000) **10**

4.1 Alternative bark conversion 10

4.2 Alternative effective second formant calculation 11

4.3 Reachable acoustic space 11

5 Testing emergence for a small community **13**

5.1 Small-community-like network 13

5.2 Community roles 14

5.3 Community behaviour 14

5.4 Updating the imitation game to work with the small community network 15

6 Results **16**

6.1 TODO 16

7 Discussion **17**

7.1 TODO 17

Extra figures **18**

References **19**

Project goal and supplied code

During the Evolution of Speech course taught at the VUB in 2021-2022 we, Computer Science students, were introduced to this multidisciplinary field by reviewing multiple important papers of the field. As the course was taught by Bart de Boer, who has an Oxford University Press published book on the origins of vowel systems and many papers in the field, we also reviewed some of de Boer's papers (de Boer, 2001; de Boer, 2000; de Boer and Thompson, 2018; de Boer and Zuidema, 2010). As a Computer Science student with limited linguistic knowledge, papers from de Boer using Agent-Based Modelling (ABM) techniques for studying phenomena in the field were found the most interesting (de Boer, 2000; de Boer and Zuidema, 2010). Because of this, we opted to extend upon de Boer (2000). The exact project goal and an overview of the supplied code are given in further detail here.

1.1 Project goal

It was chosen to re-implement and extend the paper on self-organization in vowel systems by de Boer (2000) for this project. Whilst the original C++ source code of de Boer (2000) was provided to us, his students, it was dated and not so well documented. This was to be expected as the code was not originally meant for distribution. To further ground our understanding of the paper and make extensions on this work easier, we have chosen to do a re-implementation in Python. The written code is well documented, easily extendable and most importantly, publicly available under the GPL V3 license (Bontinck, 2021; "GNU General Public License", 2007). This enables readers to not only easily reproduce the results of de Boer (2000) and the extensions provided here but also gives them a great basis for future projects. The latter was something we felt was lacking and feel is an important contribution of this work. We also addressed some of the *ad hoc* decisions in the original version. To be more precise, this report provides an alternative way of converting to the bark scale and determining the effective second formant of a produced sound. This was found to not influence the results, as is further discussed in chapter 4.

In the original version of the imitation game proposed by de Boer (2000), agent pairs are picked at random. Whilst this is an understandable simplification for his work, it made us wonder if the findings hold for more complex structures. Initially, it was considered to use scale-free networks, as we thought this would better represent a human network. However, the actual realism of scale-free networks is debated and one of our colleagues wanted to go this route already (Broido & Clauset, 2019). Because of this, we opted to model a small community consisting of agents with different roles and influences. These agents and their roles evolve over time to eventually be replaced with new agents. It is thus a more dynamic and varying setting than the one used by de Boer (2000). This is done to further ground the initial hypothesis by de Boer, namely: "The structure of vowel systems is determined by self-organization in a population under constraints of perception and production." - de Boer (2000).

1.2 Important files

Accompanied by this report is a copy of the GitHub repository created for this project (Bontinck, 2021). It includes all files needed to reproduce the experiments, including saved versions of the games used for figures and statistics in this report. An overview of the most important files is given below:

- `README.md`
  - General information of the GitHub repository with hyperlinks to important files and documentation.
- `code-output`
  - All figures generated by the provided code, some of which are used in this report.
- `code/notebooks`
  - Jupyter notebooks and plain py files going over different aspects of the code for this project.
  - `1_implementing_de_boer_2000.ipynb`: step by step re-implementation of code by de Boer (2000).
  - `imitationGameClasses.py`: all classes needed to play imitation games as specified by de Boer (2000).
  - `2_recreating_de_boer_2000.ipynb`: step by step re-collection of results by de Boer (2000).
  - `3_alternative_bark_experiments.ipynb`: step by step re-collection of results by de Boer (2000) using a less ad hoc variant of the bark converter and effective second formant weighting function.
  - `4_adding_small_communities.ipynb`: step by step implementation of the community based imitation games (extension).
  - `communityImitationGameClasses.py`: all classes needed to play the community based imitation games.
  - `5_evaluating_small_communities.ipynb`: evaluation of the community based imitation games.
- `code/html-exports` and `documentation/installation`
  - HTML export of the above discussed Jupyter notebooks, ideal for those who want to view the notebooks without installing the Anaconda environment.
  - Install instructions for the used Anaconda environment of this project (macOS and Ubuntu).

Relevant literature

In this chapter, a summary of the most important aspects of de Boer (2000) is given. We also discuss some literature on network structures in ABMs to further defend why we think our extension is a useful one.

2.1 Summary of de Boer (2000)

It was seen from previous work, such as that by Liljencrants et al. (1972), that functional properties give rise to human-like vowel systems. Such findings are often found through direct optimisation, in the case of (Liljencrants et al., 1972) this is by doing a minimisation on the potential energy to find that optimising for acoustic difference gives rise to realistic vowel systems. However, whilst such findings are great, they give rise to another question, *how* does this optimisation take place. This *how* question is one that (de Boer, 2000) tries to understand by studying an ABM playing imitation games. This paper of de Boer (2000) is inspired by his PhD thesis, which is longer and more detailed (de Boer & Steels, 1999). If the ABM, which makes use of simple self-organised agents, gives rise to human-like vowel systems, it *could* be possible that self-organisation played an important role in the evolution of speech. This could part is important, as ABMs can't directly prove a hypothesis.

The ABM described by de Boer (2000) consists of agents who play imitation games. These imitation games consist of two randomly picked agents where one is the *speaker* and the other is the *imitator*. The speaker says a random sound from his *sound repetoire*. The imitator replies with one of his known sounds that lies closest to the *perceived* one. The speaker then communicates a *non-verbal* signal to either confirm or reject the imitation to be correct. A correct imitation is one where the perceived sound is closest to the initially produced sound for the speaker. The imitator uses this information to update his sound repertoire. The sound repertoires of agents are non-fixed and initially empty. From this description it becomes clear an agent should have three important skills: a sound *production*, *perception*, and *storing* mechanism. Besides this, the agent should also be able to *learn* from the non-verbal signals. Chapter 3 re-implements this ABM, where it is described in more detail how these different components work and some of the *ad hoc* decisions made by de Boer (2000) to make it work.

Chapter 3 goes into more detail on the metrics used to evaluate the findings of this ABM. In that chapter, the found evaluation metrics are also replicated. From these found results, de Boer (2000) concludes that self-organization can explain properties found in human vowel systems. He states that one should not study a vowel system by its individual vowels (as done by Chomsky, 1991) or as a whole (Liljencrants et al., 1972) but rather concerning the population it is used in. This work by de Boer (2000) remains one of his most cited works and has proven to be influential in the field. Because of this, we found it fit to re-implement it so that newcomers can get a grasp on experiments in the field and to further defend the findings by de Boer (2000).

2.2 Importance of network structure for emergence

The extension to the above discussed ABM provided with this report is one on the network structure used. As discussed, the pair of agents to play an imitation game each iteration are chosen at random in the implementation of de Boer (2000). Given enough iterations, this will evolve from a random network to a fully connected network. This contradicts with human networks where a certain hierarchy exists and not every person will adopt his speech to another person. For example, a professor in the English language would not adopt his sound repertoire to that of a newborn. More importantly, the results of ABMs have been shown to depend heavily on the underlying communication network used, whether they are used in linguistic applications or not (Bonabeau, 2002; Petrov et al., 2021; Will et al., 2020).

To model human communities, scale-free networks are often used. Scale-free networks are networks where the degree distribution of nodes follows a power law. A preferential attachment like property in human psychology is often given as a reason that these networks are well representative of a human social network Wang et al. (2010). However, it is often debated whether such scale-free networks appear as often in nature as it is claimed to by some (Broido & Clauset, 2019).

Another commonly used network is a *small-world* network. An overview of these three network structures is given in Figure 2.1. Many other network structures exist as it is an important topic of graph theory. We believe a network that lies between small-world and scale-free networks is a good fit to represent early human communities. Hence, such a directed and weighted network is presented in chapter 5.

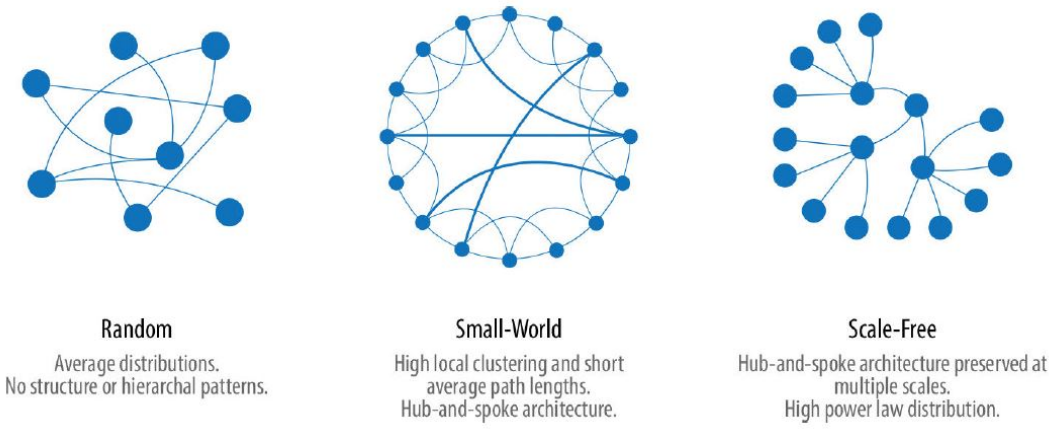


Figure 2.1: Random, small-world and scale-free networks. Figure by Needham (2019).

Re-implementing de Boer (2000)

The original C++ source code of de Boer (2000) was provided to us, de Boer's students. However, this code was dated and not meant for distribution, which made it difficult to be used or extended upon. Because we saw the value in a well documented and easy to extend implementation of this project, we decided to fully re-implement it in Python. This was done incrementally, with each step described in `1_implementing_de_boer_2000.ipynb`. The working of the code was validated by reproducing the results of the experiments by de Boer (2000) in `2_recreating_de_boer_2000.ipynb`. This chapter will summarise the development and findings of these two Jupyter notebooks. We stress that all of the discussed code in what follows is either derived from the textual description in de Boer (2000) or the provided C++ code.

3.1 Producing sounds

The ABM of de Boer (2000) consists of agents who can produce, perceive, and remember speech sounds in a human-like way. The most important component for producing sounds in a human-like manner is the articulatory synthesizer. This synthesizer takes as input three parameters related to the vocal tract configuration: the tongue position (p), the tongue height (h) and the lip rounding (r). These input parameters are represented using a `Phoneme` class object. The outputs of the synthesizer are the first four formant frequencies of the corresponding vowel: F_1 to F_4 expressed in Hz. This output is represented as a `Utterance` class object. The conversion between input and output happens based on equations given by de Boer (2000), table 2. A small typo in the provided calculation for F_3 has been corrected in our code. De Boer (2000) used interpolation from known data to create these equations. We used these known points for validating our F_1 to F_4 conversions from the p, h and r input and found them to match perfectly with the given data. It is noted that the interpolated approach of de Boer (2000) makes the synthesising process faster than a full vocal synthesizer system but forms a clear simplification.

The above-discussed process is implemented in the `Synthesizer` class. A synthesizer is initialised by providing two types of noise. The `max_noise_agent` and the `max_noise_ambient`. The agent noise is applied to the phoneme before utterance creation (Equation 3.1), the ambient noise is applied to the utterance after creation. The applied noise, λ, is uniformly picked from: $\frac{-\psi}{2} \leq \lambda \leq \frac{\psi}{2}$, with ψ being the provided parameter. ψ is one of many important parameters. In the experiments by de Boer (2000), only the ambient noise is used.

$$F_i^{agent}(p, h, r) = F_i(p + \lambda, h + \lambda, r + \lambda) \tag{3.1}$$

$$F_i^{ambient} = F_i * (1 + \lambda) \tag{3.2}$$

3.2 Perceiving sounds

With the `Synthesizer` and helper classes in place, an agent can produce a signal that represents sound in a human-like manner. For an agent to perceive these signals in a human-like manner, the `Bark Operator` class is created. This class is responsible for working with utterances. Remember that utterances were the first four formants of a generated sound, in Hertz. As the name of this class implies, the Bark scale is used by this class and is differing from the previously used Hertz scale. It represents frequencies in a manner that is closer to human perception. It goes from the four formant representation in Hertz to a two formant representation in Bark consisting of the first formant and the *effective second-formant* (F_2').

We took the conversion formulae and calculation formula for the effective second formant straight from de Boer (2000). The conversion from Hertz to Bark and back used by de Boer (2000) is again interpolated from data and given in Equation 3.3. It is also admitted by de Boer (2000) that his calculations for determining F_2' are a bit ad hoc. The used equations are given in Equation 3.4. The fourth case for determining F_2' has been changed from de Boer (2000) to reflect the effective equation used in the available code. The critical distance (c) used for calculating F_2' can be provided as an optional argument of the class instance but is set to 3.5 for all experiments. Because of the interpolated conversion and ad hoc F_2' calculations, we have also foreseen an `alternative_bark_conversion` parameter. If set to `True`, the Bark operator will use alternative methods for both of these functions. This is further discussed in chapter 4.

$$Bark = \begin{cases} \frac{ln(Hertz/271.32)}{0.1719} + 2 & Hz > 271.32 \\ \frac{Hertz-51}{110} & Hz \leq 271.32 \end{cases} \tag{3.3}$$

$$F_2' = \begin{cases} F_2 & F_3 - F_2 > c \\ \frac{(2-w_1)F_2 + w_1 F_3}{2} & F_3 - F_2 \leq c \text{ and } F_4 - F_2 > c \\ \frac{w_2 F_2 + (2-w_2)F_3}{2} - 1 & F_4 - F_2 \leq c \text{ and } F_3 - F_2 < F_4 - F_3 \\ \frac{(2-w_2)F_3 + w_2 F_4}{2} - 1 & F_4 - F_2 \leq c \text{ and } F_3 - F_2 \geq F_4 - F_3 \end{cases} \tag{3.4}$$

Having our Bark space configured, we can implement a distance measure between utterances as specified in Equation 3.5. This distance measure can be used as a way for the agent to determine the closest sound in his repertoire to the one it heard. This must happen in the Bark space as equal distances in bark correspond to roughly equal human-perceptual distances of sound. This is not the case for Hertz, as humans have a harder time differentiating higher frequencies. With this human-like distance measure, agents can now perceive sounds and compare them with their known sounds. This λ is again an important parameter for the simulations. It is set to 0.3 for all experiments in the project unless specified differently. This value is seen as realistic by de Boer (2000), Ladefoged (1985), Schwartz et al. (1997), and Vallée (1994).

$$D = \sqrt{(F_1^a - F_1^b)^2 + \lambda(F_2'^a - F_2'^b)^2} \tag{3.5}$$

3.3 Representing agents

The `Agent` class makes use of all previously discussed classes as well as the `Sound` helper class. The `Agent` class takes a synthesizer and Bark operator as arguments for initialising. It also has over ten optional parameters one of which is a logging capability handy for debugging purposes. The `Sound` class is used to store a known sound of an agent. It consists of the phoneme, utterance, usage count and success count of the sound. The utterance of this sound is determined by synthesising the provided phoneme in a noiseless environment.

To discuss the functions of an agent, it is easiest to present a typical imitation game flow. Algorithm 1 shows the actions performed by a randomly picked agent who starts an imitation game. If the agent known sound repertoire is empty a completely random vowel is inserted by picking random values between 0 and 1 for the Phoneme parameters. A different randomly picked agent plays the role of imitator and responds to the heard utterance using the process shown in Algorithm 2. If the imitator's known sounds repertoire is empty, it will add a *similar sound* to the one it heard. It does this by checking eight *corner* sounds it can produce and pick the one which is closest in distance to the heard one. Afterwards it improves this sound further by using its `improve_sound` function for the agent specific amount of times (`max_similar_sound_loops` parameter). The `improve_sound` function tries all possible permutation's of the phoneme parameters by either keeping the value or adding/substracting the agent specific step size (`phoneme_step_size` parameter).

Algorithm 1 The say_something function of an imitation game initiator agent

if No known sounds **then**
  Add random sound to known sounds
end if
$S \leftarrow$ random known sound
Update usage count of S
Remember chose of S
Return utterance of S using own bark operator

Algorithm 2 The imitate_sound function of an imitator

Require: U_{in}: the heard utterance
if No known sounds **then**
  Add similar sound for U_{in} to known sounds
end if
Remember U_{in}
Find closest known sounds S to U_{in}
Update usage count of S
Remember chose of S
Return utterance of S using own bark operator

In the second phase of the game, the initiator validates the imitation it hears in a non-verbal manner. This process is shown in Algorithm 3. The agent validates if the closest known sound to the heard imitation utterance is the sound he used to start the game. He also communicates this to the imitating agent in a non-verbal manner. He updates the success count accordingly and prepares himself for the next round. The process of preparing for a new round is given in Algorithm 4. This consists of resetting the game variables such as the *last_spoken_sound*

variable. The agent then updates its count of games played, together with the success and imitator/initiator count. Based on the agent specific `cleanup_prob`, `new_sound_prob` and `merge_prob` the agent will potentially remove bad sounds, add a semi-random new sound or merge similar sounds. A sound is thus removed periodically if its success rate is below the agent specific `sound_threshold_agent` and used at least `sound_minimum_tries`, which is also agent-specific. A sound is also added semi-randomly periodically. We call this process semi-random as multiple random vowels will be tried based on the agent specific `max_semi_random_loop`, and the sound that had the greatest summed distance will be used as the new sound. Finally, similar sounds are also merged periodically. The agent does this by validating if both the utterance or phonemes don't lie too close. Phonemes lie too close if their parameters differ less than 0.17 in total. Utterances are too close if they can't be distinguished taking into account the noise of the environment. Both of these calculations are taken from the code provided by de Boer (2000).

Algorithm 3 The validate_imitation function of an initiator

Require: U_{in}: the heard imitation utterance
  Retrieve last spoken sound S
  Find closest known sounds S' to heard utterance
  $success \leftarrow S = S'$?
  if *success* **then**
    Update success count of S
  end if
  Prepare for new game
  Return *success*

Algorithm 4 The prepare_for_new_game function of an agent

Require: *imitator*: whether or not the agent was an imitator in the played game
Require: *success*: whether or not the played game was a success
  Update agent games count
  if *success* **then**
    Update agent success count
  end if
  if *imitator* **then**
    Update agent imitator count
  else
    Update agent initiator count
  end if
  Remove bad sounds per agent-specific odd
  Merge similar sounds per agent-specific odd
  Add semi-random sound per agent-specific odd
  Reset game variables

To end a game cycle, the imitator agent will process the non-verbal imitation success communication. It does this using the process described in Algorithm 5. If the imitation was successful the agent will use the previously described `improve_sound` function once to make the spoken sound better match the heard utterance. If the imitation was not successful and the used sound has a success ratio lower than the agent specific `sound_threshold_game`, the sound is also improved as described before. However, if the success ratio of the sound is above this threshold it is assumed that the spoken sound is a correct imitation of other sounds in the network and thus a similar sound is added to the one heard as a reaction. The process of adding this similar sound is identical as described when the sound repertoire of an imitator was empty.

Algorithm 5 The process_non_verbal_imitation_confirmation function of an imitator

Require: *success*: whether or not the imitation was a success
  Update agent games count
  if *success* **then**
    Update agent success count
    Improve used sound to better match the heard utterance
  else if Low success ratio of spoken sound **then**
    Improve used sound to better match the heard utterance
  else
    Add a similar sound to the heard utterance
  end if
  Remove bad sounds per agent-specific odd
  Merge similar sounds per agent-specific odd
  Add semi-random sound per agent-specific odd
  Reset game variables

3.4 Playing and analysing games

To play the imitation game for a specified amount of iterations and to store the results, two additional classes were made. The `GameState` class stores a copy of the agents at a certain point and the iteration number at which the copy was made. It also has a plot function to plot the sound repertoires of agents, grouped per agent. The `GameEngine` contains a simple loop to play imitation games for a specified amount of iterations. The `play_imitation_game` function returns a list of `GameState` objects at the specified `checkpoints` (iteration numbers) by playing the imitation games.

To make evaluating the obtained `GameState` objects, and thus experiment results, easier, a `Statistics` class is created. It allows for easy calculation of the average and standard deviation for *energy*, sound repertoire *size*, and *success rates* of agents. These results can also easily be plotted by the provided functions. To compare the emerged sound systems with real human vowel systems, a function `plot_known_vowels_over_sounds` is provided. This plots the known vowels used to derive the interpolated synthesizer function on top of a plot from the agents' vowel repertoires.

The data and plots obtainable from these classes are used in chapter 6 to discuss the results.

Resolving some ad hoc decisions from de Boer (2000)

In the previous chapter, we discussed the re-implementation of the imitation games as proposed by de Boer (2000). We mentioned that both the vocal synthesizer and bark conversion were interpolated from publicly available data. The calculations for determining effective second formant weight (F_2') were found rather ad hoc and definitions varied between the paper and the available code.

We think the design decision to model the vocal tract from interpolated data is understandable in this context. Not only is it faster than modelling a more complex synthesizer, but the fact that 18 different vowels were used for the interpolation makes it a reasonable approximation. Especially when considering far fewer sounds were learned on average by the agents due to the limited acoustic space, we think the interpolated vocal tract doesn't impose a conflict for the study. However, the interpolated bark conversion and calculations for F_2' were something we think could influence the results. For this reason, this chapter will go into detail on how we tested variants for these calculations using the alternative bark operator.

4.1 Alternative bark conversion

When discussing the `Bark Operator` class in section 3.2 we mentioned the inclusion of the optional `alternative_bark_conversion` parameter. When this parameter is set to `True` both the conversion from Hertz to bark and back as well as the F_2' calculations will differ from those described in Equation 3.3 and 3.4 respectively.

It was found that there is no single definition for the bark scale. After comparing multiple variants we decided to settle for the bark conversion used by MatLab[1]. The conversion from Hertz to bark is given in Equation 4.1. The inverse can analytically be derived and was tested in the `2_recreating_de_boer_2000.ipynb` notebook.

$$\begin{aligned} \texttt{intermediate} &= \frac{26.81 * Hz}{1960 + Hz} - 0.53 \\ bark &= \begin{cases} \texttt{intermediate} + (0.15 * (2 - \texttt{intermediate})) & \text{if } bark < 2 \\ \texttt{intermediate} + (0.22 * (\texttt{intermediate} - 20.1)) & \text{if } bark > 20.1 \\ \texttt{intermediate} & \text{else} \end{cases} \end{aligned} \tag{4.1}$$

[1] https://nl.mathworks.com/help/audio/ref/hz2bark.html

4.2 Alternative effective second formant calculation

For the calculation of the effective second formant weight (F_2'), de Boer (2000) stated that the strategy used by Schwartz et al. (1997) would probably be better. For that exact reason, we implemented the strategy proposed by Schwartz et al. (1997). This strategy consists of two parts, determining a value c and calculating F_2' based on that c. The process to determine c is described in Figure 4.1. After c is determined, F_2' can be calculated. This is done by using Equation 4.2. It is noted that Schwartz et al. (1997) used yet another bark conversion method.

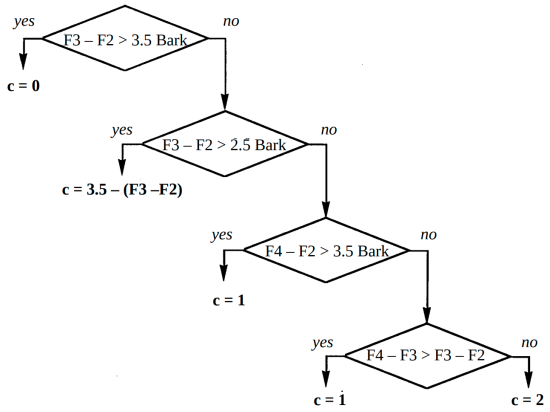


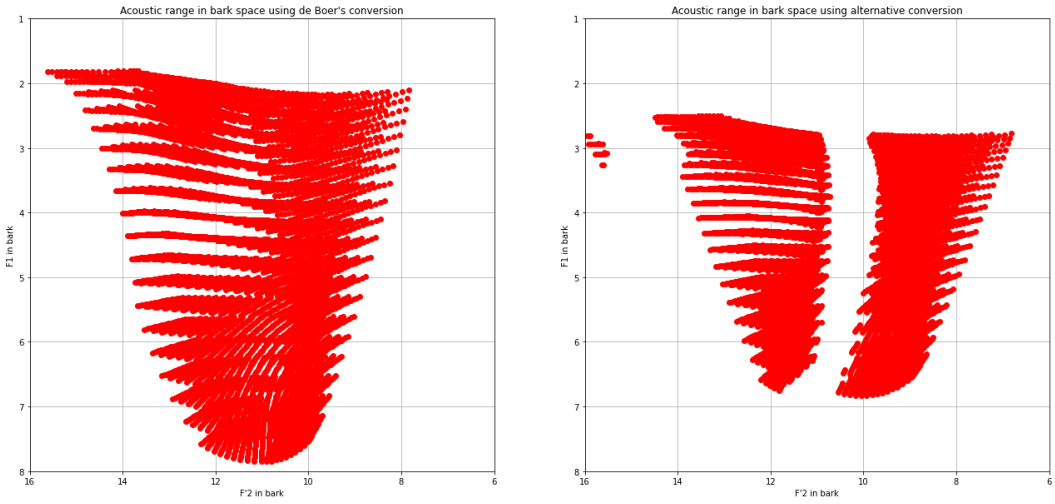
Figure 4.1: C calculation for effective second formant equation. Strategy and figure by Schwartz et al. (1997)

.

$$F_2' = \frac{c_2F_2 + c_3F_3 + c_4F_4}{c_2 + c_3 + c_4} \begin{cases} c_2 = 1, c_3 = 0, c_4 = 0 & \text{if } c = 0 \\ c_2 = 1, c_3 = 0.5, c_4 = 0 & \text{if } 0 \leq c \leq 1 \\ c_2 = 0, c_3 = 1, c_4 = 0.5 & \text{else} \end{cases} \tag{4.2}$$

4.3 Reachable acoustic space

To visualise the impact of the change `Bark Operator` the reachable acoustic space was approximated by plotting multiple points. This experiment is visualised in Figure 4.2. The alternative implementation visually differs in multiple ways. As computer scientists, we find the more continuous nature of the implementation by de Boer (2000) more pleasing. However, an expert should ideally determine which of the two is more realistic. In Chapter 6 it is discussed that the results of the experiment remain similar independent of the used variant.



(a) Default implementation (b) Alternative implementation

Figure 4.2: Reachable acoustic space of both Bark Operator alternatives.

Testing emergence for a small community

In the previous chapter we addressed some *ad hoc* design decisions made by de Boer (2000). In this chapter, we present another extension on the original model by de Boer (2000). Instead of using random selection for initiator and imitator pairs, we use a small-community-like network. This network is based on 5 main groups of agents, each with different influences on the other. The network can behave both vertical and horizontal, i.e. generational or not. This extension is a demonstration of how the provided code can be used to easily configure new experiments. This implementation is discussed step-by-step in the `4_adding_small_communities` notebook.

5.1 Small-community-like network

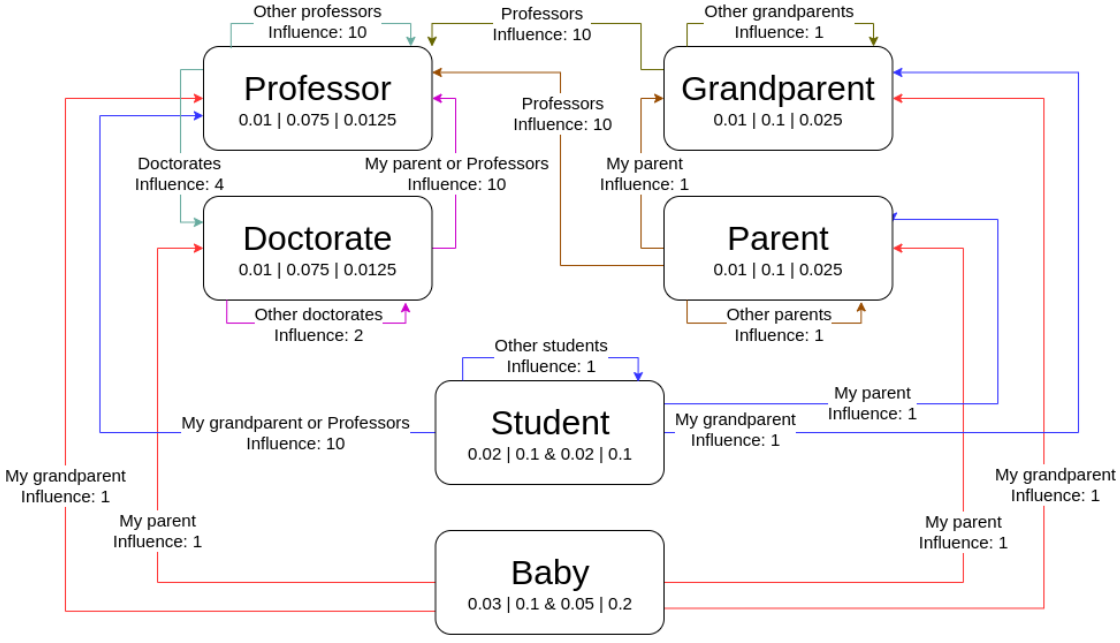


Figure 5.1: Properties of small community network used.
Arrows indicate an influential role for the agent accompanied by its weight.
Notation underneath role: new sound probability | ambient noise | phoneme step size.

5.2 Community roles

In Figure 5.1, a general overview of the used network is shown. From this diagram, it is visible that there are six different agent roles. These roles are stored as an `Enum` under the `Community Role` class. An agent will be one of these roles for a specified number of iterations. This number is provided as the `category_age_width` parameter for the `Community Game Engine` discussed further down this report. It is important to clarify that `Babies` and `students` don't exist together, rather in the first half of the provided age width parameter these agents are babies, and in the second half they evolve to become students. Each agent has at most one unique parent, depending on if that agent is still alive or not. The parent agent might also still have a parent, resulting in a grandparent for the youngest agent. After the number of iterations is passed specified by the age width parameters, agents *shift up* in this network and their role evolves. That means that students become either doctorates or regular parents, determined by their own parent's role. Doctorates become professors and parents become grandparents. Professors and grandparents get replaced by new agents who start with an empty vowel repertoire. These agents are all children of one unique parent agent or doctorate agent. This keeps the distribution constant throughout the games.

It is noted that the role names were chosen to be intuitively comprehensible. In the case of primitive humans, a similar structure was likely to exist, with professors being leaders of tribes for example.

5.3 Community behaviour

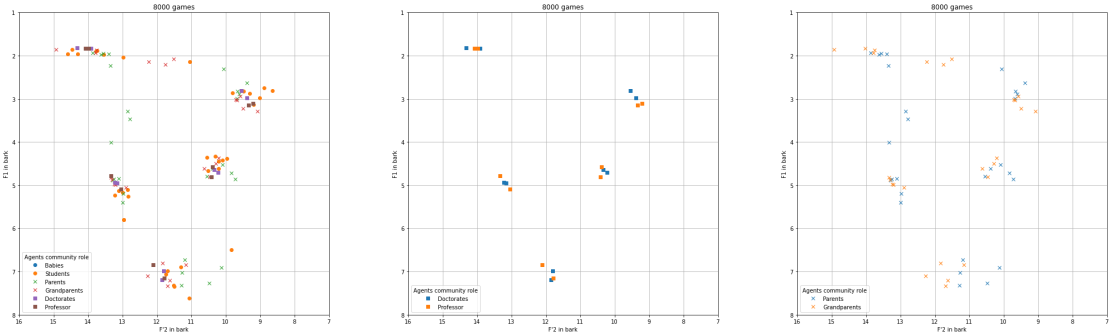
Besides a `Community Role`, an agent also has a `Community Behaviour`. This behaviour describes multiple things:

- The probability of adding a new sound. This is shown as the first number under the role name in the diagram.
- The synthesizer to be used. This allows specification of the ambient noise and optional agent noise to be used as described in section 3.1. The noise levels are given as the second number under the role name in the diagram. If two values are given, the second describes the agent noise.
- The phoneme step size as described in section 3.3. This is shown as the third number under the role name in the diagram.
- The roles that an agent is influenced by. This will be respected when choosing an initiator when this agent is an imitator. This is shown as arrows in the diagram.
- The strength of the above-described influence. This integer number represents the number of optimisation steps taken after playing a game with an agent of that role. It is given as the label of the arrows on the diagram.

5.4 Updating the imitation game to work with the small community network

Updating the classes discussed in Chapter 3 to work with the above described small community network and behaviour is relatively straightforward. Due to the use of Python classes, it is possible to create a child class where functions can be re-implemented only if needed for the wished extension. A nice example of this is the `Community Agent` class. It inherits from the regular Agent class described in 3.3 and takes the community role and behaviour as extra initialisation parameters. By overwriting the `improve_sound` function, a simple loop calling the original `improve_sound` function can be made. This loop iterates for a specified amount depending on the opponent's role influence strength. This mechanism is thus the one to make the influence by agents of a certain role bigger than the others. The opponent's role can be stored by creating an additional `prepare_current_game` function which is called before the game is played. Likewise, a `change_agent_role_and_behaviour` function can be added to evolve the agent to its new form. These simple changes are everything that is needed to adopt the agent class to work in the described network.

To call these additional functions and to evolve the agents on the correct iteration, the `Game Engine` was also updated to a `Community Game Engine` similarly. A `Community Game State` and `Community Statistics` class was also created to provide additional plotting functionalities. This consist of being able to plot and evaluate only the highly educated (professor and doctorate) or the regular educated (parent and grandparent) agents. Figure 5.2 shows an example of these plotting functionalities.



(a) All agents (b) Highly educated agents (c) Regular educated agents

Figure 5.2: Different plotting functionalities using the same game state object.

Results

As discussed during in the re-implementation chapter, some of the design decisions by de Boer (2000) were rather ad hoc.

6.1 TODO

TODO

Discussion

TODO

7.1 TODO

TODO

Extra figures

To make the report more readable some figures are not provided directly in the text. These figures are provided here.

References

Bonabeau, E. (2002). Agent-based modeling: Methods and techniques for simulating human systems. *Proceedings of the National Academy of Sciences*, *99*(suppl 3), 7280–7287. https://doi.org/10.1073/pnas.082080899

Bontinck, L. (2021). *Eos project @ vub 2021 - 2022* [GitHub commit: TODO...]. Retrieved December 27, 2021, from https://github.com/pikawika/eos

Broido, A. D., & Clauset, A. (2019). Scale-free networks are rare. *Nature Communications*, *10*(1). https://doi.org/10.1038/s41467-019-08746-5

Chomsky, N. (1991). *The sound pattern of english*. MIT Press.

de Boer, B. (2001). *The origins of vowel systems.* [Oxford: Oxford University Press The book based on my PhD. thesis.]. Oxford University Press.

De Smet, R. (2020). *Vub latex huisstijl* [GitHub commit: d91f55...]. Retrieved November 2, 2020, from https://gitlab.com/rubdos/texlive-vub

de Boer, B. (2000). Self-organization in vowel systems. *Journal of Phonetics*, *28*(4), 441–465. https://doi.org/10.1006/jpho.2000.0125

de Boer, B., & Steels, L. (1999). Self-organization in vowel systems. https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.32.5947&rep=rep1&type=pdf

de Boer, B., & Thompson, B. (2018). Biology-culture co-evolution in finite populations. *Scientific Reports*, *8*(1). https://doi.org/10.1038/s41598-017-18928-0

de Boer, B., & Zuidema, W. (2010). Multi-agent simulations of the evolution of combinatorial phonology. *Adaptive Behavior*, *18*(2), 141–154. https://doi.org/10.1177/1059712309345789

Gnu general public license. (2007, June 29). *Free Software Foundation*. http://www.gnu.org/licenses/gpl.html

Ladefoged, P. (1985). *Phonetic linguistics : Essays in honor of peter ladefoged*. Academic Press.

Liljencrants, J., Lindblom, B., & Lindblom, B. (1972). Numerical simulation of vowel quality systems: The role of perceptual contrast. *Language*, *48*(4), 839. https://doi.org/10.2307/411991

Needham, M. (2019). *Graph algorithms : Practical examples in apache spark and neo4j*. O'Reilly.

Petrov, A., Akhremenko, A., Zheglov, S., & Kruchinskaia, E. (2021). Is network structure important for protest mobilization? findings from agent-based modeling. *The monitoring of public opinion economic and social changes*. https://doi.org/10.14515/monitoring.2021.6.2021

Schwartz, J.-L., Boë, L.-J., Vallée, N., & Abry, C. (1997). The dispersion-focalization theory of vowel systems. *Journal of Phonetics*, *25*(3), 255–286. https://doi.org/10.1006/jpho.1997.0043

Vallée, N. (1994). *Systèmes vocaliques : De la typologie aux prédictions* (Doctoral dissertation). Université Stendhal.

Wang, M.-Y., Yu, G., & Yu, D.-R. (2010). The scale-free model for citation network. *2010 IEEE International Conference on Intelligent Computing and Intelligent Systems*, *1*, 773–776. https://doi.org/10.1109/ICICISYS.2010.5658791

Will, M., Groeneveld, J., Frank, K., & Müller, B. (2020). Combining social network analysis and agent-based modelling to explore dynamics of human interaction: A review. *Socio-Environmental Systems Modelling*, *2*, 16325. https://doi.org/10.18174/sesmo.2020a16325