



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

<b>1</b>	<b>Project goal and supplied code</b>	<b>1</b>
1.1	Project goal . . . . .	1
1.2	Important files . . . . .	2
<b>2</b>	<b>Relevant literature</b>	<b>3</b>
2.1	Summary of de Boer (2000) . . . . .	3
2.2	Importance of network structure for emergence . . . . .	4
<b>3</b>	<b>Re-implementing de Boer (2000)</b>	<b>5</b>
3.1	Producing sounds . . . . .	5
3.2	Perceiving sounds . . . . .	6
3.3	Representing agents . . . . .	6
3.4	Playing and analysing games . . . . .	8
3.5	Improving de Boer (2000) . . . . .	8
<b>4</b>	<b>Testing emergence for a small community</b>	<b>10</b>
4.1	TODO . . . . .	10
<b>5</b>	<b>Results</b>	<b>11</b>
5.1	TODO . . . . .	11
<b>6</b>	<b>Discussion</b>	<b>12</b>
6.1	TODO . . . . .	12
	<b>Extra figures</b>	<b>13</b>
	<b>References</b>	<b>14</b>

# 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 3.5.

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 extension provided by this project.
  - `5.evaluating_small_communities.ipynb`: evaluation of the created extension with respect to de Boer (2000).
- **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 repertoire*. 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 4.

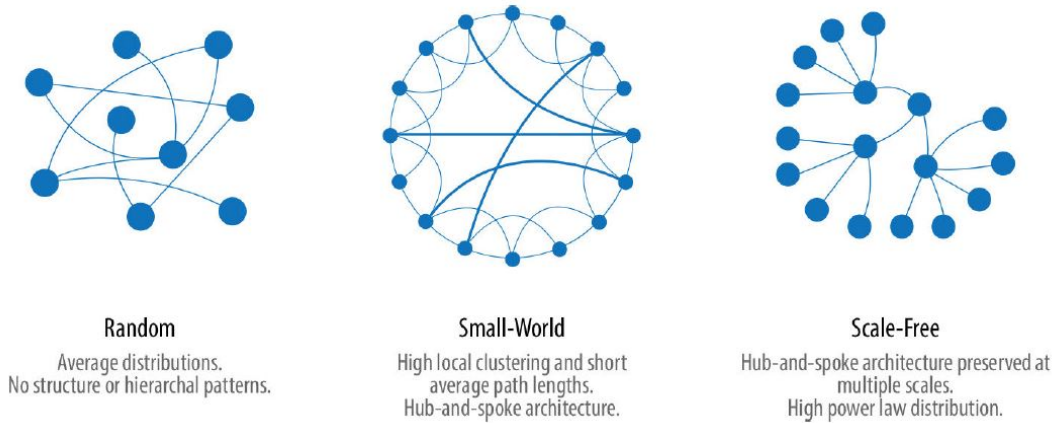


Figure 2.1: Random, small-world and scale-free networks. Figure by Needham (2019).

# 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 textual description in de Boer (2000) or the provided C++ code.

## 3.1 Producing sounds

de Boer (2000) system 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 vowel parameters: the tongue position ( $p$ ), the tongue height ( $h$ ) and the lip rounding ( $r$ ). The outputs of the synthesizer are the first four formant frequencies of the corresponding vowel:  $F_1$  to  $F_4$  expressed in  $Hz$ . The conversion between input and output happens based on the synthesizer equations given by de Boer (2000), table 2. 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.

From the above-described synthesizer equations, we can make the **Synthesizer** class. To represent input and output formally, we make use of two helper classes: **Utterance** and **Phoneme**. The former stores an utterance consisting of the four formants  $F_1$  to  $F_4$ . This is the output of the **synthesise** function. A phoneme is used as input for the same function and stores the  $p$ ,  $h$  and  $r$  parameters. Two types of noise can be assigned to a **Synthesizer** object. 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,  $\lambda$ , is uniformly picked from:  $\frac{-\psi}{2} \leq \lambda \leq \frac{\psi}{2}$ , with  $\psi$  being the provided parameter.  $\psi$  is one of many important parameters. In the experiments by de Boer (2000), only the ambient noise is used.

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

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

### 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. It is also admitted by de Boer (2000) that his calculations for determining ( $F'_2$ ) are a bit ad hoc. The critical distance used for calculating ( $F'_2$ ) can be provided as an optional argument. 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 section 3.5.

Having our Bark space configured, we can implement a distance measure between utterances as specified in Equation 3.3. 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  $\lambda$  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).

$$D = \sqrt{(F_1^a - F_1^b)^2 + \lambda(F_2'^a - F_2'^b)^2} \quad (3.3)$$

### 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 response 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 picking the one which is closest in distance. 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/subtracting 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 heard utterance
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, as well as the success and imitator/initiator count respectively. 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 on a periodic basis. 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 new sound. Finally, similar sounds are also merged on a periodic basis. 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 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 reaction. The process of adding this similar sound is identical as described when the sound repertoire of an imitator was empty.

### 3.4 Playing and analysing games

TODO

### 3.5 Improving de Boer (2000)

TODO

---

**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 heard utterance

**else if** Low success ratio of spoken sound **then**

Improve used sound to better match 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

---

# Testing emergence for a small community

TODO

## 4.1 TODO

TODO

# Results

TODO

## 5.1 TODO

TODO

# Discussion

TODO

## 6.1 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>