TELKOMNIKA, Vol.x, No.x, 2020, pp.101~10x
ISSN: 1693-6930, accredited First Grade by Kemenristekdikti, Decree No: 21/E/KPT/2018
DOI: 10.12928/TELKOMNIKA.vxix.xxxx

Automatic sound synthesis using the Fly algorithm

Zainab A. Abbood[1], Imad S. Alshawi*[2], Asaad A. Alhijaj[2], Franck P. Vidal[3]
[1] Department of Pathological Analysis Science, College of Science
[2] Department of Computer. College of Computer Science and Information Technology
[1,2] University of Basrah, Basrah, IRAQ
[3] School of Computer Science, Bangor University, UK
*emadalshawi@gmail.com

Abstract

Our study is demonstrated a new type of evolutionary sound synthesis method. This work based on the Fly algorithm, a Cooperative Co-evolution algorithm; it is derived from the Parisian evolution approach. The algorithm has relatively amended the position of individuals (the Flies) represented by 3-D points. The Fly algorithm has successfully investigated in different applications, starting with a real-time stereo vision for robotics. Also, the algorithm shows promising results in tomography to reconstruct 3-D images. The final application of the Fly algorithm was generating artistic images, such as digital mosaics. In all these applications, the flies' representation started for simple, 3-D points, to complex one, the structure of 9-elements. Our method follows evolutionary digital art with the Fly algorithm in representing the pattern of the flies. They represented in a way of having their structure. This structure includes position, color, rotation angle, and size. Our algorithm has the benefit of graphics processing units (GPUs) to generate the sound waveform using the modern OpenGL Shading Language.

Keywords*: cooperative co-evolution, digital sound, evolutionary algorithm, fly algorithm, Parisian evolution, waveform.*

1. Introduction

Nowadays, computers are used to simulate different types of human functions in the form of digital representation. For example, sound synthesizers are commonly run in the digital computer by implementing a computer program that produces digital sound samples (waveform)[1]. The algorithm is designed to emulate the target sound by tunning certain internal parameters[2]. These parameters are variable determined by the user implementation according to the desired sound. The algorithms that generate sound are called Sound synthesis techniques (SSTs). The classic SSTs implemented following a traditional mathematical technique for emulating the internal parameters. The estimated parameters for designing a functional form of SSTs are dependent problems relying on human skills. However, generating sounds is a remarkably difficult task, it is slightly inharmonic and the partials process a certain stochastic low-amplitude, high-frequency deviation[3].

Traditional sound synthesis needs comprehensive human experiences[4] and along with refinement, processes to estimate the internal parameters. The motivation behind this research is to find an alternative method to synthesis the target sound. Usually, techniques rely on tuning a large number of parameters, up to 200 or more in some cases, mathematically. However, the synthesizers show that they can be non-linearly responded to some parameters. This means any change of one parameter might be effected to another and small changing on one parameter can cause a large change in the sound[5]. Currently, these parameters have been automatically expressed through solving SST as an optimization problem using Artificial Evolution (AE)[1].

The first attempt to generate complex music sounds using the evolutionary paradigm described by Dawkins,1986[6].

In 1993, Horner *et al.*[7] presented a type of evolutionary method which is Genetic algorithms (GAs) to estimate the internal parameter of FM synthesizers. Wehn[8] proposed automatic digital synthesizer circuits. The circuits are generated sounds comparable to a sampled (target) relying on a GA. This algorithm initials the population of the elements

Received , 2020; Revised, 2020; Accepted, 2020

(individuals) of the circuit and successively improves these elements to get a better circuit for generating the target sound.

The next section demonstrates the previous works of sound synthesis. This section focuses on the problems that close to our approach. Section 2 shows a general review of the “Parisian evolution" strategy in specific the Fly algorithm. Section 3 represents the adaptation of the Fly algorithm into evolutionary sound synthesis. However, the Fly algorithm previously used for medical tomography reconstruction, robotic and digital art generator applications. This section follows the result. Finally, the remarkable conclusion gives in the last section.

2. Evolutionary Sound Synthesis

Sound synthesis has been active research for more than four decades. Several techniques for sound synthesizers like additive synthesis, subtractive synthesis, frequency modulation, wavetable synthesis or physical modeling[9]. Sound synthesis from the image has several implementations; one possible application could be expressed using an artist to draw sound[10].

Bragand *et al*.[10] presents a user interface to generate sound from image relying on a Voronoi algorithm. Another approach, Photosounder[11] the authors apply an inverse Fast Fourier transform (FFT) on the input image to generate sound after they examine the image as a magnitude spectrogram.

Sound synthesis implementations have manipulated using an evolutionary algorithm (EA). In 2001, Garcia and his colleague applied genetic programming (GP) and perceptual distance metrics for measuring the distance between the target and produced sounds[12].

The researchers in [13] enhance the FM synthesis model with some changes. They rely on more waveforms than a sine wave, like a sawtooth wave. For synthesis, this paper uses GA with a fitness function that weighted sum of two spectrum-comparison metrics. The crossover selection parameter has shown a significant effect on the problem domain.

Yong [14] follows Lai and his colleagues for sound synthesis using GA. However, he depends on the DFM synthesis model with the same type of fitness function. The implementation is done by MatLab to produce an output spectrum according to input (target) sound spectrum file.

Other researchers [15] have been used method which is cellular automata (CA). This technique can be classified as one of a class of evolutionary algorithms for modeling dynamic systems that modify several characteristics with time.

Our work follows the evolutionary sound paradigm. The sound synthesis solved by EA as an optimization problem. In specific, our proposed method relies on the Fly algorithm, which is a type of Cooperative Co-evolution (CoCo) strategy[16].

In this research, we treat sound synthesis as image reconstruction. The principles of our method follow the case of studying sound synthesis as a specific case of the set cover problem by placing a group of tiles as a set on a square shape region to convergent a colored image[17]. Here sound synthesis falls under the subject of generating digital mosaic by fitting mosaic tiles on a surface. Each tile needs setting up 9 elements (3 color components, height, width, rotation angle, and 3-D position), the search space is complex which has 9_N dimensions. Such a problem is a difficult optimization problem to solve. In this research, we propose to solve this problem using a type of Cooperative Co-evolution Algorithm (CCEA) called “Parisian evolution"[18]. The Parisian approach is differing from classical EAs. For the solution of the optimization problem, the EAs are searching on the best one individual as the solution. While the CCEA strategy looks on a set or subset of individuals of the population as a final solution. This means that every individual is a part of the solution, and all individuals collaborating to build the final solution[19].

3. Overview of the Fly Algorithm

For solving the sound reconstruction problem, we follow the main mechanics as in Parisian evolution. Figure 1 shows the principles of the steady-state of the Parisian evolution algorithm[20].

This algorithm similar to classical EA, which contains the usual genetic operators of an EA: selection, mutation, and recombination as well as the additional ingredients two types of fitness, as the following:

- *Global fitness* evaluated the whole population[21] [22].

- *Local fitness* evaluated the single contribution of the individual to the global solution.

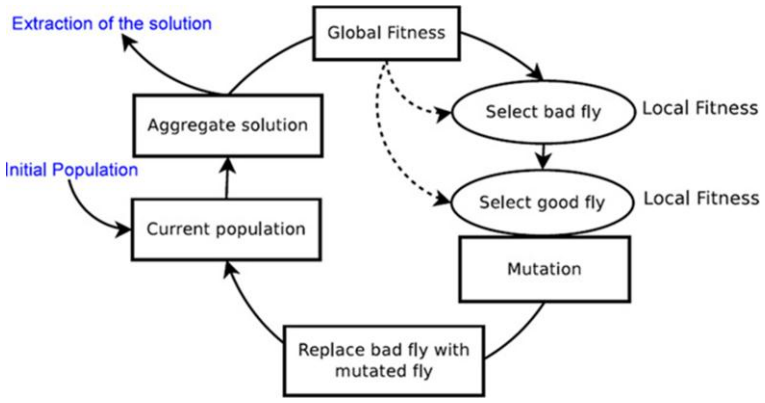

Figure 1. Steady-state Cooperative Co-evolution strategy.

In Section 2, we mentioned that our work depended on the Parisian (Fly) strategy. The individuals in this algorithm correspond to two types of structures. One corresponds to exceedingly simple primitives: The flies [23] represent a 3-D position only. The other structure contains 9-elements (see Figure 2):

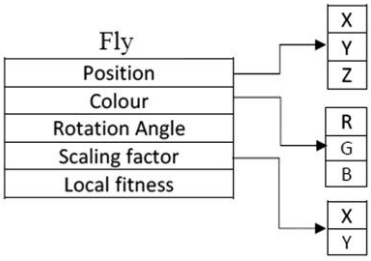

Figure 2. The fly data structure

- **The position** is a vector of 3D coordinates (*x, y, z*), which is randomly generated between 0 and *width* -1, 0 and *height* 1, and 1 and -1 respectively (with *width* and *height* the number of pixels in the image along the *x*- and *y-axis*). An example of an image generated by an initial population has shown in Figure 3.
- **Colour** represented by three components (*r, g, b*) which are red, green and blue respectively. The color elements are randomly generated between 0 and 1 for precise displaying various colors for each individual.
- **Rotation Angle** is randomly produced to cover a complete angle between 0 and 360.
- **The scaling factor** dominates the size of the tile over the two-dimensional coordinate system through the horizontal and vertical axis.
- **Local fitness** scales its marginal contribution against the global solution.

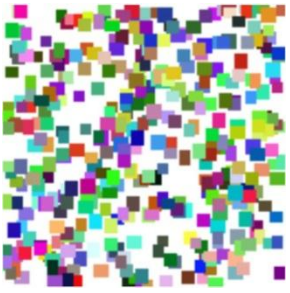

Figure 3. Random initial population

The goal of our algorithm is to optimize the 9 elements of all individuals. The proposed method aims to minimize the global fitness function. We depend on the sum of absolute error (SAE) which called Manhattan distance too [24]. This scale assesses how is a good the population toward the reference sound.

$$SAE(RCI, INI) = \sum_i \sum_j |INI(i,j) - RCI(i,j)| \quad (1)$$

The SAE is compared the reference image INI with the reconstructed image pop.

While assessing the performance of a single y, we use local fitness. This fitness knows “marginal fitness", *Fm(i)* (see equation 2). Our algorithm looks to improve the population performance by increasing the good flies against the bad flies. The SAE metrics plus the leave-one-out cross-validation method are used to gauge the degree of compatibility of Fly *i* to the reference image. We select a fly that has a good contribution to the population and leaves out the bad one.

$$Fm(i) = SAE(RCI - \{i\}, INI) - SAE(RCI, INI) \quad (2)$$

With *RCI* – {*i*} the image calculated with all individuals except Fly *i*. The sign of the value of *Fm (i)* referred to as a different interpretation:

- sgn($F_m(i)$) becomes less than 0 when the difference (error) is greater with Fly *i*. This means that the Fly *i* is incompatible with the optimal solution for all the rest of individuals.
- sgn($F_m(i)$) becomes greater than 0 when the difference (error) is less with Fly *i*. This means that the Fly *i* is converged to the optimal solution.
- sgn($F_m(i)$) becomes equal to 0 when the difference (error) is the same with Fly *i*. This means that the Fly *i* is not valuable nor destructive.

As the algorithm processing going on as the bad flies decrease. For the selection stage, we use the Threshold selection operator [25]. If $F_m(i) \leq 0$, then Fly *i* can be left out; else it is a good candidate for reproduction. For stopping criteria of the algorithm, we depend on the algorithm struggle to find bad flies to kill.

4. Results

In this section, we design automated sound synthesis through the Fly algorithm and perceptual distance metrics to measure the distance between reference (*ref*) and generated (*gen*) sound. The previous section shows this work relying on SAE matrices to quantify the error between the input and target sound.

For fast processing, we compute the SAE with the help of a Graphics Processing Unit (GPU) using the OpenGL Shading Language (GLSL) (see Figure 4).

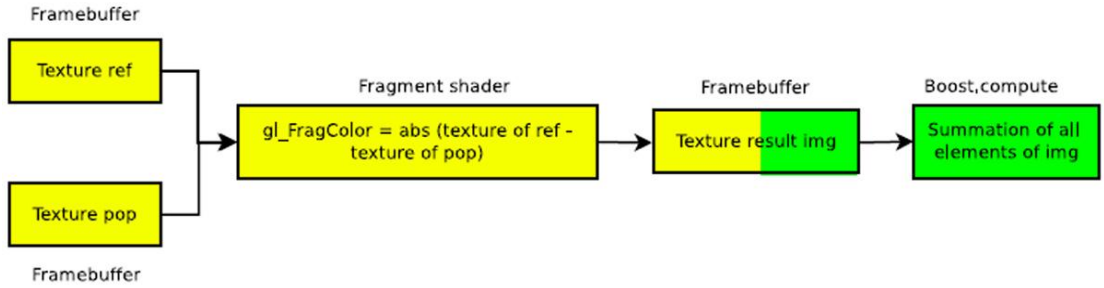

Figure 4. Computation of the marginal fitness on GPU for two images using GLSL and OpenGL (in yellow and green boxes respectively)

The *gen* sound is generated offline using a Frame Buffer Object (FBO). Sounds including *ref* and *gen* are stored using 2-D OpenGL textures. The next step is that the pixel-wise absolute error between (*ref*) and (*gen*) is computed after the texture is passed to a GLSL shader program. The process of summation is completed on the GPU with the help of the OpenGL application of the reduction operator provided by Boost Compute [26]. It efficiently supplies the SAE.

Also, our method uses only an operator of mutation for generating a good fly and replace with a bad fly during an iteration of the optimization technique. Crossover operator excluded to ensure that produced recent fly partially modified from the good previous generation fly. In other words, when we use crossover maybe two good flies producing a new fly in between are very likely to be bad.

Our method is tested with parameters showing in Table 1 and Table 2, using 9 x number of flies-D search space. In the first steps of the algorithm, the flies randomly generated using the square tiles (see Figure 3). The mutation probability is fixed to 100 % due to the crossover is not suitable in our algorithm. Our practical part of the algorithm processed automatically. However, the image (and its size) and the number of individuals are selected by the user. To avoid slow down the whole process, the user balanced the number of flies with image size for avoiding premature convergence.

Table 1. Parameters for the first testing in the algorithm

Parameter	Value
Image size	200 x 400
no. files	3500
no. generations	200000
probability of mutation (p_m)	100%
probability of crossover	0.0%

Table 2. Parameters for the second testing in the algorithm

Parameter	Value
Image size	200 x 400
no. files	7500
no. generations	200000
probability of mutation (p_m)	100%
probability of crossover	0.0%

For realistic sound synthesis, our algorithm replaces the square tiles with a stripe mask (see Figures 5 and 6).

We use a *twisted wave* sound editor for reading sound. The Mono sound system used in this research. A waveform sampled at 8000 Hz. Figure 7 shows an example of a reference sound which is used in this article.

Figure 5. Mask using stripline

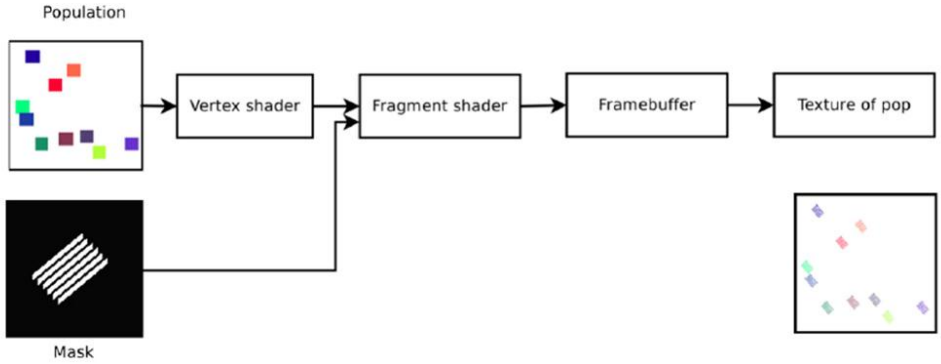

Figure 6. Fitting stripe line on square tiles

Figure 7. Reference image

The result gradually builds through the Fly algorithm iteration as we mention started with random stripe Flies. Then, this Fly reconstructs to reach to the target image.

Results of Figures 8 and 9 rely on Table 1 and Table 2 respectively. Figures 8 and 9 show how flies reconstruction gathering around the reference image, as the algorithm executes as the shape getting sharper. The algorithm stops either the error getting smaller between the reference and reconstructed image or the algorithm relying on stopping conditions. However, the result of Figure 9 shows the converging of the reconstructed image with the reference image more than the result of Figure 8. We conclude that as the number of Flies increases as the realistic result you got in the end.

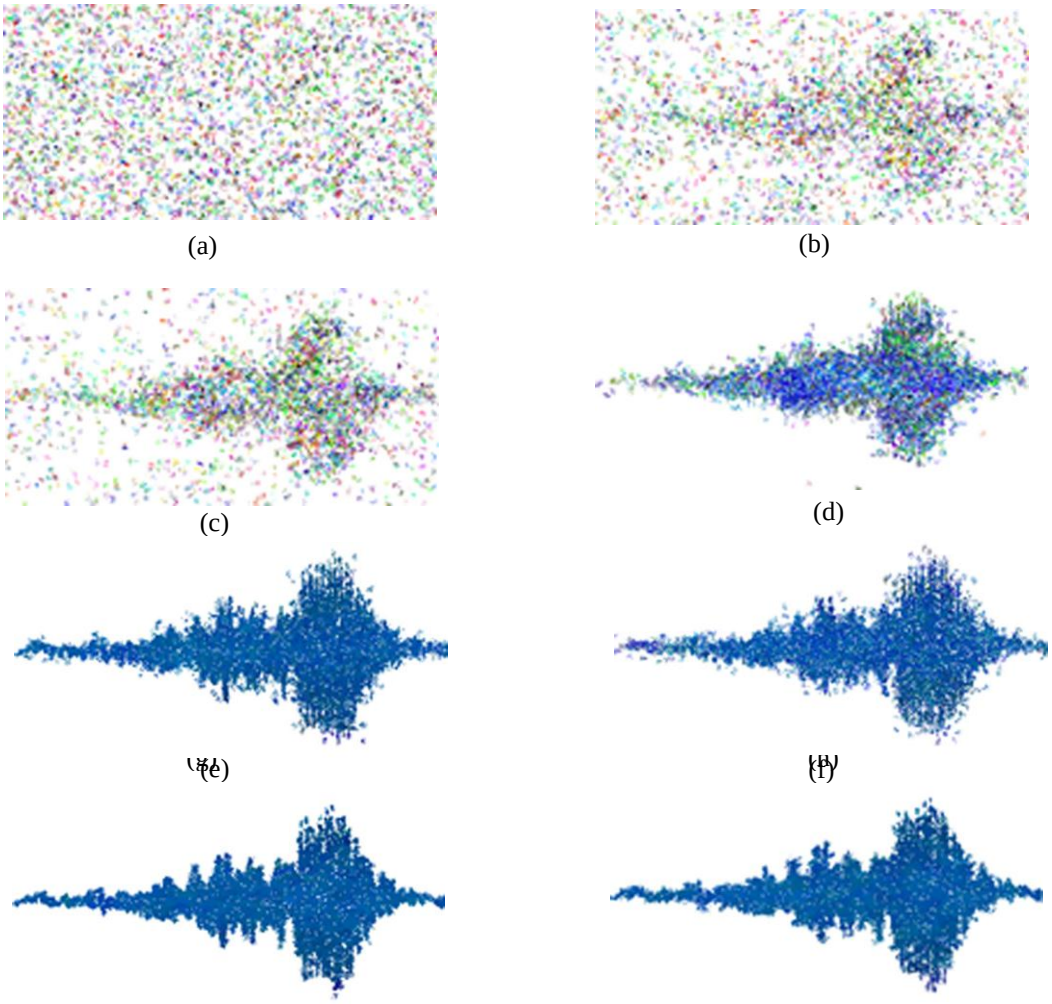

Figure 8: Results of sound reconstruction using the Fly algorithm depending on the parameters of Table 1.

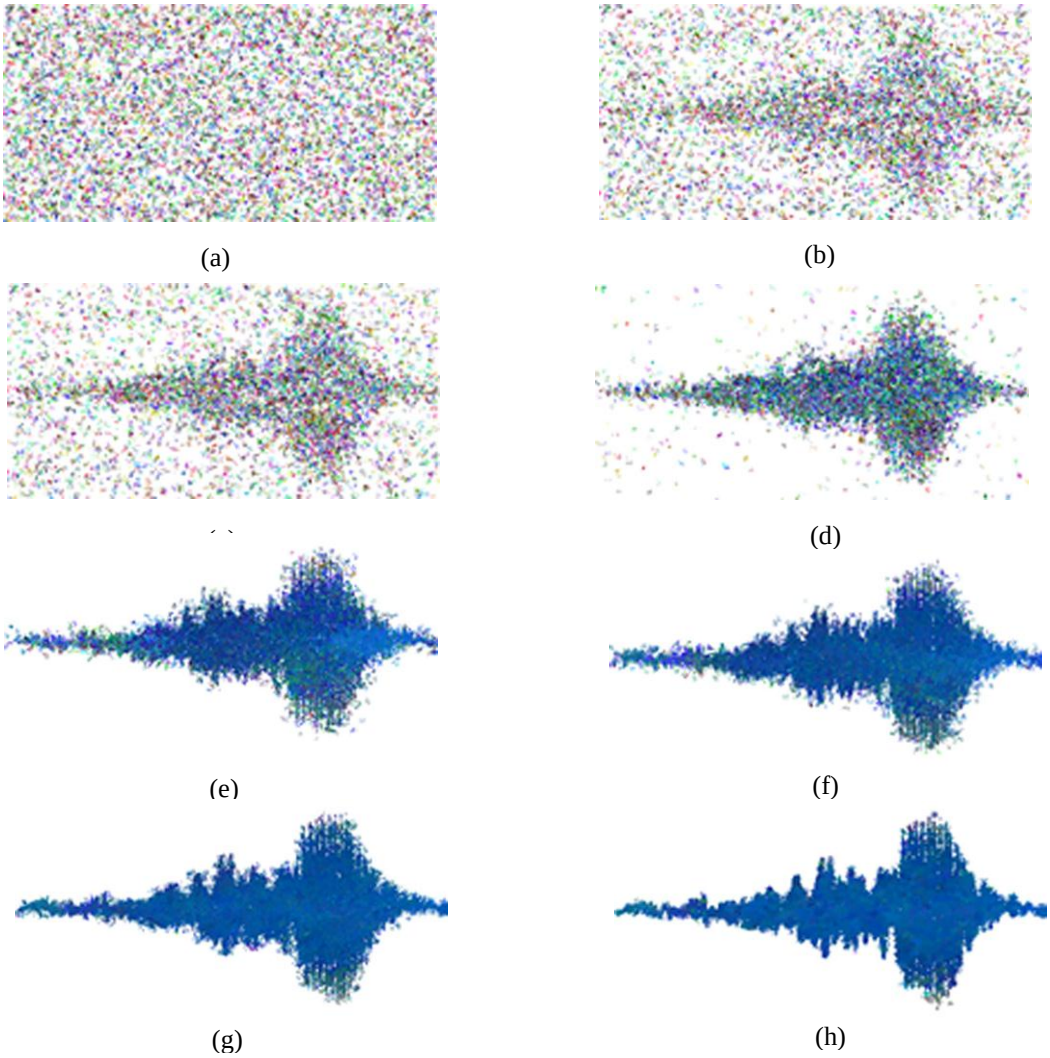

Figure 9: Results of sound reconstruction using the Fly algorithm depending on the parameters of Table 2.

5. Conclusion

The proposed method is tackled the problem in the field of Evolutionary sound. The method addresses the problem as an image reconstructions algorithm. The algorithm depends on hybrid techniques that inherited from AE, scientific computing and Computer Graphics (CG). The AE represents the Fly algorithm for reconstructing the sound template. For generating the sound data, real-time CG rendering and Graphics Processing Unit (GPU) are used to compute the fitness function. The algorithm uses stripe tiles to create sound visual effects.

References

[1] R. A. Garcia, "Automating the design of sound synthesis techniques using evolutionary methods," in *COST G-6 Conference on Digital Audio Effects, Limerick, Ireland*, 2001.
[2] J. Riionheimo and V. J. E. J. o. A. i. S. P. Välimäki, "Parameter estimation of a plucked string synthesis model using a genetic algorithm with perceptual fitness calculation," vol. 2003, p. 758284, 2003.
[3] M. F. Caetano, J. Manzolli, and F. J. Von Zuben, "Interactive Control of Evolution Applied to Sound Synthesis," in *FLAIRS Conference*, 2005, pp. 51-56.
[4] P. Dahlstedt, "Evolution in creative sound design," in *Evolutionary computer music*, ed: Springer, 2007, pp. 79-99.
[5] J. McDermott, N. J. Griffith, and M. O'Neill, "Evolutionary computation applied to sound synthesis," in *The Art of Artificial Evolution*, ed: Springer, 2008, pp. 81-101.

[6] R. Dawkins, *The blind watchmaker: Why the evidence of evolution reveals a universe without design*: WW Norton & Company, 1996.
[7] A. Horner, J. Beauchamp, and L. J. C. M. J. Haken, "Machine tongues XVI: Genetic algorithms and their application to FM matching synthesis," vol. 17, pp. 17-29, 1993.
[8] K. Wehn, "Using ideas from natural selection to evolve synthesized sounds," in *Proceedings of Digital Audio Effects workshop*, 1998.
[9] E. Miranda, *Computer sound design: synthesis techniques and programming*: Taylor & Francis, 2012.
[10] V. Bragard, T. Pellegrini, and J. Pinquier, "Pyc2Sound: a Python tool to convert images into sound," in *Proceedings of the Audio Mostly 2015 on Interaction With Sound*, ed, 2015, pp. 1-4.
[11] L. K. Photosounder. (2019). *photosounder*. Available: http://photosounder.com
[12] J. Manzolli, A. Maia Jr, J. Fornari, and F. J. N. R. Damiani, "A Method for Sound Synthesis Based on Genetic Algorithms," pp. 40-49, 2013.
[13] Y. Lai, S.-K. Jeng, D.-T. Liu, and Y.-C. Liu, "Automated optimization of parameters for FM sound synthesis with genetic algorithms," in *International Workshop on Computer Music and Audio Technology*, 2006, p. 205.
[14] Y. S., "Automatic FM Synthesis Based on Genetic Algorithm," *Music Technology, McGill University,* 2007.
[15] J. Serquera and E. R. Miranda, "Evolutionary sound synthesis: rendering spectrograms from cellular automata histograms," in *European Conference on the Applications of Evolutionary Computation*, 2010, pp. 381-390.
[16] J. Louchet, "Stereo analysis using individual evolution strategy," in *Proceedings 15th International Conference on Pattern Recognition. ICPR-2000*, 2000, pp. 908-911.
[17] R. M. Karp, "Reducibility among combinatorial problems," in *Complexity of computer computations*, ed: Springer, 1972, pp. 85-103.
[18] A. M. Boumaza and J. Louchet, "Dynamic flies: Using real-time parisian evolution in robotics," in *Workshops on Applications of Evolutionary Computation*, 2001, pp. 288-297.
[19] C. Vo, L. Panait, and S. Luke, "Cooperative coevolution and univariate estimation of distribution algorithms," in *Proceedings of the tenth ACM SIGEVO workshop on Foundations of genetic algorithms*, 2009, pp. 141-150.
[20] F. P. Vidal, D. Lazaro-Ponthus, S. Legoupil, J. Louchet, É. Lutton, and J.-M. Rocchisani, "Artificial evolution for 3D PET reconstruction," in *International Conference on Artificial Evolution (Evolution Artificielle)*, 2009, pp. 37-48.
[21] L. Panait, S. Luke, and J. F. Harrison, "Archive-based cooperative coevolutionary algorithms," in *Proceedings of the 8th annual conference on Genetic and evolutionary computation*, 2006, pp. 345-352.
[22] E. Sapin, J. Louchet, and E. Lutton, "The Fly Algorithm Revisited-Adaptation to CMOS Image Sensors," in *IJCCI*, 2009, pp. 224-229.
[23] Z. A. Abbood, O. Amlal, and F. P. Vidal, "Evolutionary art using the fly algorithm," in *European Conference on the Applications of Evolutionary Computation*, 2017, pp. 455-470.
[24] F. Al Farid, A. Ghods, M. Shohag Barman, A. Shah, and J. Uddin, "An Efficient Algorithm for Accelerating Image Restoration Using Euclidean and Manhattan Distances."
[25] F. P. Vidal, J. Louchet, J.-M. Rocchisani, and É. Lutton, "New genetic operators in the fly algorithm: application to medical PET image reconstruction," in *European Conference on the Applications of Evolutionary Computation*, 2010, pp. 292-301.
[26] K. Lutz. (2015). *Boost. compute*. Available: http://boostorg.github.io/compute/.