Flora Generation and Evolution Algorithm for Virtual Environments

Carlos Mora, Sandra Jardim
Smart Cities Research Center
Polytechnic Institute of Tomar
Tomar, Portugal

Jorge Valente,
Undergraduate Student
Polytechnic Institute of Tomar
Tomar, Portugal

Abstract — **A crucial aspect to game development is the ability to establish the player emersion in a simulated and virtual world, presenting engaging content and maximizing the gaming experience. The demand for new and hand-customized content keeps increasing, in playable maps that are ever expanding in size, without compromising quality or differentiation. The usage of Procedural Content Generation (PCG) allows computers to generate game content and produce distinguishable and unique instances amongst the ones generated allowing to better replicate reality and deliver intricate systems without the costs and time consumption of human intervention.**

As a key element for the appeal of many games, natural life exhibits both complex and distinguishable behaviors, from species to species as well as from individual to individual. Biodiversity is maintained by processes operating over broader spatial and temporal scales, and as such, simulating plant diversity is a process better suited to operate under PCG algorithms.

In this study, we describe a procedural flora evolution algorithm that can replicate a flora species life cycle for a practical 3D game setting (Universe 51), using scientific knowledge to better reproduce such intricate behaviors. We based some part our approach on complex intelligence algorithms, as well as in some original processes targeted at our project specificities. The evolution algorithm starts with the end results of a flora generation algorithm that can create millions of different flora species based on selected biological parameters. This data is then interpreted by the evolutionary algorithm, that uses these values to determine each species survival chances in the surrounding environment. Our results were able to replicate and autonomously manage plant species life cycle, under a dynamic environment with potentially varying conditions.

Keywords – procedural content generation; flora generation algorithm; flora evolution algorithm; natural life simulation.

I. INTRODUCTION

Some natural phenomena, such as the survival of living entities and the success of some species in each environment, rely on Optimization and Search to overcome the constraints that these environments impose on them [1]. This complex optimization strategies make for compelling problems and are one of the reasons of why some algorithms are inspired by nature [1, 2].

Using Procedural Content Generation is an increasingly popular method to develop game content. Vegetation can be used in many games for a realistic and thus immersive look. The presence of vegetation is not merely aesthetic; it may serve as hiding place, as raw material for inventive players, to direct a player towards a certain direction or reference the climate surrounding the player and lead to changes in gameplay [3].

In this paper, we researched some of the most relevant information available, and best practices on Procedural Generation Technics, to develop a Flora Generation and Evolution Algorithm, to be applied on a real-world application, an in-development game named Universe 51, that allows for the exploration of +25 million planets in photo quality graphics. To do so, we opted to separate the procedure into two main algorithms: the first would handle flora species generation, and insert the data into an SQL database, and the second would use the generated species to populate an existing world, control how individual flora specimens would behave in the surrounding environment as well as propagate and colonize new areas, simulating the natural processes with real scientific basis.

II. BASE ASSUMPTIONS

A. Flora characteristics, dispersal, and Fitness

In Nature, species survival often depends on their ability to adapt in various ways, whether to find food quickly or avoid predation. These are typically Optimization/Search problems that give their offspring the best chance to survive and thrive [1].

A major pressuring force in species survival is competition, which can be either between different species (interspecific) or between plants of the same species (intraspecific). To evaluate and understand competition [4], studies in the past two decades, have concluded that intraspecific competition appears to be significantly stronger than interspecific competition for most pairs of co-occurring species. Additionally, in ~30% of the cases analyzed, the effect of intraspecific competition was negative and the interspecific effect positive, a situation which promotes coexistence between plants of different species. Cases in which both inter and intraspecific effects were positive (~1%), or in which the intraspecific effect was facilitative, but the interspecific effect was competitive (~1%), were significantly rare [4].

As plants have a variety of modes to spread seeds, many authors accept a classification into autochory and allochory [5]. Autochory refers to plants that spread by themselves, and allochory means the plants spread through external forces [2, 3]. While it has become widely accepted that most plant

species are dispersed by more than one dispersal vector or mode [5, 6], for the scope of this paper we will consider only Autochory, as the software where the algorithm will be applied does not presently include animals, wind sources or other external forces that we could try to replicate in the algorithm.

Studies have found there are significant differences in dispersal related to different environmental and topographic conditions. For example, wind dispersal is more effective in an alpine landscape, with heavy turbulences, compared to lowland conditions [7]. Altitude has been known to be a significant conditioner in many life aspects of flora species [8, 9]. As altitude increases, the climate shifts toward more stressful conditions for plant growth: lower mean temperatures, higher precipitation, longer snow cover and, thus, a shorter growing season, lower atmospheric pressure and higher solar radiation that induces high temperatures at the ground level [8, 9].

Considering those factors, we introduced Altitude as a limiting filter to the plant species generated, such as restricting taller plants to lower altitude habitats. Studies observed that trees cannot grow at high altitude, because of cold temperature or lack of available moisture. This would somewhat mimic the real world as plants that share the same habitat often exhibit similar characteristics (i.e., morphology, reproduction, diaspore dispersal) [10], generally identified as biological traits. This kind of ‘‘filter’’ phenomenon occurs in nature and removes the species that lack the necessary biological traits [9, 10].

Some species have also developed different strategies to deal with specific survival optimization problems [1]. The temperature surrounding the plant is known to affect the rate of plant growth and development, with each species having a specific temperature range comprising of a minimum, maximum, and optimum temperature [11]. One other factor that can limit plant growth is the concentration of CO2, but since the pool of atmospheric CO2 is so large and mixed, plants do not heavily compete for CO2 in our planet [12]. Using this kind of limitations can improve development of game assets based on natural processes, enabling developers with more levers of control to differentiate between different planets or realities, creating a sense of novelty and discovery.

B. Procedural generation in games

Computer games are increasingly present in our modern day lives, with hundreds of millions of players joining daily and around the world, to play a variety of titles [3]. In 2020, according to the US Entertainment Software Association (ESA), 75% of all U.S. households have at least one person who plays while, in total, 64% of all U.S. adults and 70% of those under 18 regularly play video games [13]. In the context of games and simulations, complex and realistic virtual worlds gained more and more importance [14, 15].

In contrast to traditional methods of content production, Procedural Content Generation for games (PCG) consists of the application of computers algorithms to generate game content, distinguishing between instances generated, and generating entertaining instances for the players [3]. As a definition, [16] classified procedural content generation as “any kind of automatically generated asset based on a limited set of user-defined input parameters”, sometimes also referred as amplification algorithms. Usage of PCG techniques has been around the game industry, in various forms, for decades [14].

Using PCG enables making complex game worlds, with limited time, without putting a large burden on the game content designers [3, 14]. Additionally, Procedural content may increase the replayability of a game, by changing levels or quests and offering new experiences in each new session [14, 15], effectively extending the life span of a title. PCG has also influenced the creation of “serious games”, which converge a learning experience in addition to the entertainment value in classic games. Their value in education comes from the changing environments that enable students not only repeat knowledge they have learned, but also apply their abilities to new and changing situations [15].

One aspect to retain is the tradeoff that occurs when generating procedural content. When an algorithm returns a particular outcome (e.g., a natural-looking forest), to be able to generate more variations of assets, textures in higher resolution, more detailed meshes or a denser planting, it inevitably requires more processing power, more memory and/or more storage [15]. For this reason, and depending on the desired outcome, the developer has to establish a trade between either performance or realism, to reach the optimal outcome for the given system or the requirements for the virtual world [15]. Generating most types of game content necessitates from a computer not only computational power, but also to access the metadata and technical values of the generated instances [3].

Considering all this, and since virtual worlds may require millions of square meters, it is indispensable to establish a way to optimize performance for the rendering process. One possible approach is described in [17] with the usage of Levels of Details (LOD), which should be used to add more terrain detail in important and frequently visited areas of a terrain that are close to the user, and to reduce detail in less important regions, e.g., in far mountain areas.

Lastly, Random Number Generators (RNG) can be used to produce a deterministic and periodic sequence of (pseudo) randomized numbers. While their functionality is referenced in many publications dealing with PCG, not all researchers agree with their omnipresence in PCG, with some authors defending that pure random generation would result in chaos [15].

C. Complex intelligence algorithms:

In scientific and engineering areas, some problems require the search for an optimal solution given a large and mutating space. For these complex, nonlinear, or discrete optimization challenges, existing traditional optimization algorithms, such as Newton’s method and the Gradient Descent method, may have a hard time finding a solution [2, 18]. For such cases, some widely used intelligence algorithms include the particle swarm optimization (PSO) algorithm [19], artificial bee colony (ABC) algorithm [20], the flower pollination algorithm (FPA) [21], and the Plant Propagation Algorithm (PPA) [1] provide examples of some of the best solutions available.

Swarm intelligence algorithms, such as ABC and PSO, are based on the interaction, communication, and cooperation of organisms in individuals of a group. While behavior and intelligence of everyone is somewhat limited and simple, it can

produce valuable overall capacity by interaction and cooperation on the biological group [2, 22]. Different studies on this area have been developed recently, focusing as well on practical problems that are required to satisfy multiple objectives, with conflicting natures towards each other, using Multi-Objective Optimization algorithms [22].

We based some of our approach on the Artificial Flora (AF) algorithm, as described in [2]. In developing the AF algorithm, its authors took inspiration in the reproduction and the migration of flora. In their study, the principle is that original plants spread seeds in a possible radius around, with the propagation distance considering the previous original plants.

In plant communities, competition for resources has been associated with generating stress for plants, and to be important for determining species distributions, as well as their evolution [12]. The three main classes of resources that limit plant growth are nutrients, water, and light, which are all resources for which individual plants compete. Soil properties are affected by numerous different nutrients, which limit plant growth in different ways [12].

An environmental fitness variable can be used to determine whether the seeds can survive or not, and consequently, if the offspring plant cannot adapt to the environment, it will die. If a seed survives, it will become original plants and spread seeds [2]. Using these principles, through multi-generational propagation, the flora will migrate to a more suitable area, completing the task of finding the optimal growth environment through the evolution, extinction, and rebirth of individual flora specimens.

III. METHODS AND DEVELOPMENT

A. Flora Generation Algorithm

The starting point of the study was to create an algorithm that, based on selected biological parameters, could generate any desired number of new species, all different from one another and insert their data into an SQL database. This database contains all the pertinent biological data of the species, such as: the optimal temperature and altitude, the preferred gas or gases, the prohibited gas or gases, the exclusive radius an individual would take, the maximal width and height for the species, the longevity, the average reproduction cycle, as well as, for visualization purposes, information about the materials composing the species model.

Since we are dealing with a potential large volume of new and alien species, we chose to base the attribution of these parameters by generating random numbers, and limiting their extremes with known and existing examples, to maintain some degree of fidelity and familiarity with the player. Additionally, the number generators were coupled with different functions, that change the frequency of values to a pattern more closely resembling for our reality. As an example, when determining the optimal temperature, the algorithm is weighted to generate less values close to the extremes.

As this algorithm is executed, some variables (preferred and prohibited gas, and reproduction cycle) are attributed in the beginning of the process, as they are the most difficult to extrapolate from plant examples on our planet. After this, a plant type is randomly established (from algae, moss, grass, shrub, or tree), with the remaining variables being generated (height, width, longevity, optimal temperature, optimal altitude, etc.) loosely based on what is seen on our planet. This allows us to create new and different species, while still maintaining elements that hopefully are still recognizable and familiar to players. By connecting characteristics like height and optimal altitude to the type of plant, we can also maintain the filter effect from altitude, that was mentioned in the introduction.

B. Flora Evolution Algorithm

After a planet is generated, a variable set of existing species will be assigned to populate it from the established database. For each species assigned, a new flora actor, with the corresponding biological values, will be randomly located in the map.

From the Flora Generation Algorithm), we can extract several variables that will determine the species physiological characteristics. Some of these have a direct effect on the in-game plants, being directly translated into the assets as soon as they are created. such as:

- Max width, Max Height and Exclusive radius – assigned to define the plant mesh/model.
- Longevity – used to establish the Actor's life span.
- Reproduction Cycle – applied directly to the algorithm.
- Flora Type – constrains the areas for initial spawning.

These are variables that have a direct effect on the plant species progression/regression in the different "ecosystems" observed in each distinct planet. Other species characteristics are used to express their physiological needs, having a direct correlation to the species presence or absence in the current world, and as such, are used to determine the species survival fitness. Specifically:

- Optimal temperature.
- Optimal altitude.
- Preferred atmospheric gas percentage.
- Prohibited atmospheric gas percentage.

All these variables will affect the species survivability, and as so are considered when determining the fitness equation for each different plant species.

In [2], the authors establish the individual fitness F using the equation:

$$F = \left| \sqrt{\frac{F(P'_{i,j\times m})}{f_{max}}} \right| \times Q_x^{(j\times m-1)} \quad (1)$$

where $F(P'_{i,j\times m})$ is the fitness for plant offspring in position j, at interaction i, with m representing the number of seeds that one plant can propagate in each interaction, f_{max} is the maximum fitness of the flora in the current generation, and $Q_x^{(j\times m-1)}$ is a value between 0 and 1 where Q_x is the selective probability.

Since in this paper is determined fitness based on the distance between the offspring and the original plant, we opted to adapt this equation by changing how to determine the fitness of current solution using the variables mentioned previously (temperature, altitude, preferred gas percentage and prohibited gas percentage). Since these variables vary wildly in order of greatness (altitudes can vary in the thousands while gas percentages at maximum vary in the tens), we decided to apply some conversions to keep the potential score consistent.

Ultimately, the equation utilized to calculate the fitness score F is:

$$F = \left| \sqrt{\frac{f_{max} - (\Delta t + 0{,}05\Delta a + 2{,}25W_g + 2B_g)}{f_{max}}} \right| \times rand(0,1) \quad (2)$$

where f_{max} is the maximum fitness of the flora in the current generation, Δt is the difference between the surrounding environmental temperature and the species Optimal Temperature; Δa is the difference between the local altitude and the species Optimal Altitude; W_g is the prohibited gas excess (difference between the max percentage tolerated and the local percentage of the prohibited atmospheric gas), B_g is the preferred gas deficit (difference between the local percentage of the preferred atmospheric gas and the minimal percentage required) and rand(0,1) denotes the independent uniformly distributed number in (0,1).

Lastly, since our algorithm operates under the principle that only successful offspring are generated, we shifted the selective probability on the equation for a randomized element, that still varies between 0 and 1, and which ensures that the survivability does not rely strictly on the value of the fitness. This allowed us to emulate that different specimens have each its own fitness score, and even when in proximity they may have different "environmental" pressures.

After estimating the fitness, each plant will generate descendancy using the same propagation method as in [2], at a distance D, from the original plant, utilizing the equation:

$$D = d_1 \times rand(0,1) + d_2 \times rand(0,1) \quad (3)$$

Where d_1 is the propagation distance of grandparent plant, d_2 is the propagation distance of the parent plant and rand(0,1) denotes the independent uniformly distributed number in (0,1).

IV. RESULTS

The full study workflow consists in generating a desired number of new species using the Flora Generation Algorithm, thus creating a new set of species available to be used from that point forward. The higher the number of species desired, the bigger will be the response time necessary to generate all the required information, with an average time obtained of around 11 seconds for every 5000 species generated.

Using then, the Flora Evolution Algorithm, we can simulate a simplified life cycle for each specimen of each generated plant species, using both information from the generation algorithm and information from the location of each specimen, regardless of any player interactions.

This way, the flora evolution algorithm works completely independent from the player, managing the processes of growth, reproduction, decay, and death, of each individual specimen spawned into the world, independently from each other. All the processes are determined according to the fitness variable, calculated using the species and the environmental characteristics. The following figure illustrates how some of the described process develop in a life- cycle for a single plant specimen:

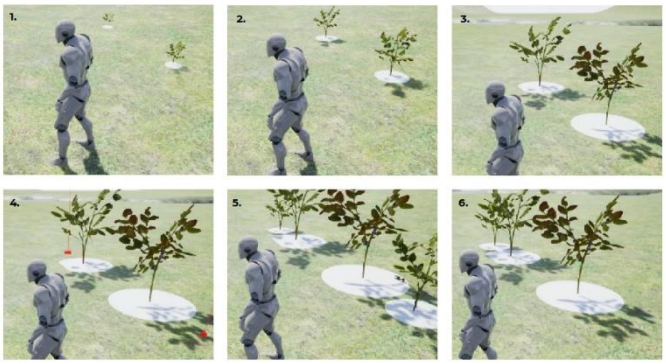

Figure 1. Plant specimen progression, through different phases of development.

From the image above we can observe a few different events:

1. Two healthy plants, shortly after generation.
2. Plants after a few cycles of growth.
3. Plants near their max size. The one closest to the player changed its model aspect to indicate a fitness reduction.
4. The furthest plant (healthy) entered the propagation phase and originated two new individuals (red indicators).
5. The new plants grow.
6. One of the newest plants entered the exclusive radius (white circle) of another and was ultimately eliminated through competition.

$\alpha + \beta = \chi$ (1) (1)

The full process starts when plant specimens are generated, with a shortened height (depending on the established growth rate) and developing further through time if the conditions are suitable for the species. During repeated time cycles, the algorithm estimates the plant fitness, using (1) and oversees the growth and development of the individual in relation to the environment. As the plant and its model grows, so does the exclusive radius (observable only during development, as a white circle under the plant).

If the conditions are suitable, the plant only stops growing once it reaches its maximum height and width. From this point on it will still re-access its fitness and reproduce periodically, as well as potentially changing the model depending on its condition. Certain game events, such as collisions between the exclusive radius of different specimens are meant to simulate competition events, which will trigger further reductions on the overall fitness. Once either it reaches its life span or the

environment stops being suitable, the plant will die and disappear. Other adaptations were also introduced to verify that plants are generated correctly in different environments.

To reduce response times, the Flora Evolution Algorithm was design so that interaction cycles do not need to be sequential allowing for jumps in interactions although subordinated to time spans. This allows to reach average response times of 30 seconds to simulate a population of 1000 individual specimens over a 25-earth year period.

V. CONCLUSIONS

Overall, we succeeded in developing an automated flora behavior algorithm that can control the species full life cycle, from generation, through propagation, and finally death. While some factors had to be adapted due to the specificities of the algorithms and databases created, we fulfilled the main objectives of this study by developing a complex procedural algorithm with real life scientific basis.

By using weighted values when creating new species data in the database, we may have introduced some bias to the results and artificially limit the variations of flora that will be represented. However, and since we are dealing with hypothetical species for a game setting, occurring in extraterrestrial planets, we decided to prioritize player satisfaction and familiarity against potential inaccuracies or limitations. Additionally, since we opted to generate these parameters and store them in a database, we can change and tune the established rules however necessary, and gradually delete or alter the species information, without having to change the game files or impacting its performance.

Since the algorithms developed had objectives and restrictions to account for, we needed a tailored algorithm that would take the characteristics of the worlds and plant species into account. This meant we needed to adapt from existing studies. While we based some of our methods on Swarm intelligence algorithms, such as [2] our approach and requirements led us to depart somewhat from the group communication centric behavior, that is used on such approaches. Since our project settings revolved around individual plant survivability, in different environments and planets, our algorithms focus different project constraints, which might not be as easily reproduced by strictly group behavior mechanisms.

One possible weakness from the algorithm developed is that most of the studies found, and used as foundation, tended to focus on inland plant species. This means that the algorithm might be less suitable to replicate aquatic plant behaviors. An interesting prospect would be to analyze the differences from aquatic and intertidal species and change the algorithm to better simulate these types of plants. By developing a more generic algorithm to create many plant species, some of the specificities that different plant types have will, inevitably, not be accurately replicated. However, doing so would certainly increase the amount of information utilized and stored, and increase the performance costs, that would be necessary to process all this data.

For further work, one important addition to the species generator algorithm would be the implementation of a growth-rate variable generated independently for each species. This might be an important feature to help visually differentiate similar species. While there is such a variable in the in-game algorithm, helping regulate and improve the behavior algorithm, it might be also beneficial to implement this from the start, under the species generator in the database.

More importantly, further work might be required to adapt and improve the performance costs of applying this algorithm in a large and widespread setting. This type of algorithm, developed in our study, might be best suited for application in larger plant species, for once because these are more easily seen by the players, meaning we would be targeting the assets performance towards the most visible. Additionally, considering that a square kilometer of forest can represent millions of plants, hundreds of thousands of small trees, and numerous small scrubs, representing plant compositions is extremely difficult [23], and minimizing the performance costs towards the most suitable game-objects should be beneficial [15].

Lastly, a more refined system to manage the game-models of all the specimens, in each different development state, would be an interesting follow-up step, which could not be achieve in this study mainly due to time restraints. If we managed to apply PCG techniques to generate the game assets, it could noticeably reduce the required performance to run these types of algorithms in a larger scale.

ACKNOWLEDGMENT

This work has been funded by national funds through FCT - Fundação para a Ciência e a Tecnologia, I.P., under the Project UIDB/05567/2020.

REFERENCES

[1] Sulaiman, M.; Salhi, A.; Fraga, E.S.; Mashwani, W.K.; Rashidi, M.M. (2016). A novel plant propagation algorithm: modifications and implementation. Sci. Int. (Lahore), 28(1):201-209.

[2] Cheng, L., Wu, X., Wang, Y. (2018). Artificial Flora (AF) Optimization Algorithm. Applied Sciences, 8 329: 1-22.

[3] Hendrikx, M.; Meijer, S.; Vann-der-Velden, J.; Iosup, A. (2011) Procedural Content Generation for Games: A Survey. ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM), 9(1), 1-22.

[4] Adler, P.B.; Smull, D.; Beard, K.H.; Choi, R.T.; Furniss, T.; Kulmatiski, A.; Meiners, J.M.; Tredennick, A.T.; Veblen, K.E. (2018). Competition and coexistence in plant communities: intraspecific competition is stronger than interspecific competition. Ecology Letters, 21: 1319-1329.

[5] Hintze, C.; Heydel, F.; Hoppe, C.; Cunze, S.; König, A.; Tackenberg, O. (2013). D3: The Dispersal and Diaspore Database – Baseline data and statistics on seed dispersal. Perspectives in Plant Ecology, Evolution and Systematics, 15: 180-192.

[6] Poschlod, P.; Tackenberg, O.; Bonn, S. (2005). Plant dispersal potential and its relation to species frequency and coexistence. In van der Maarel, E. (Ed.), Vegetation Ecology. Blackwell, pp. 68–76.

[7] Tackenberg, O.; Stöcklin, J. (2008). Wind dispersal of alpine plant species: a comparison with lowland species. J. Veg. Sci., 19: 109–118.

[8] Körner, C. (2003). Alpine plant life. Springer, Berlin, 2nd edition.

[9] Pellissier, L.; Fournier, B.; Guisan, A.; Vittoz, P. (2010). Plant traits co-vary with altitude in grasslands and forests in the European Alps. Plant Ecol., 211: 351–365.

[10] Knevel, I.C.; Bekker, R.M.; Bakker, J.P.; Klyer, M. (2003). Life-history traits of the Northwest European flora: the LEDA database. J. Veg. Sci., 14: 611–614

[11] Hatfield, J.L. & Prueger, J.H (2015). Temperature extremes: Effect on plant growth and development. Weather and Climate Extremes, 10:4-10.

[12] Craine, J.M. & Dybzinski, R. (2013). Mechanisms of plant competition for nutrients, water and light. Functional Ecology, 27: 833–840.

[13] ESA. (2020). Essential facts about the video game industry. Entertainment Software Association (ESA). Annual Report, 2020 edition. [Online] Available at: http://www.theesa.com.

[14] Shaker, N.; Yannakakis, G.; Togelius, J. (2010): Towards Automatic Personalized Content Generation for Platform Games. Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, Stanford. 63'- 68.

[15] Freiknecht, J. & Effelsberg, W. (2017). A Survey on the Procedural Generation of Virtual Worlds. Multimodal Technologies and Interaction, 1, 27.

[16] Smelik, R.M., Tutenel, T., de Kraker, K.J., Bidarra, R. (2011). A declarative approach to procedural modeling of virtual worlds. Comput. Graph. 35, 352–363.

[17] M. Duchaineau, M. Wolinsky, D. E. Sigeti, M. C. Miller, C. Aldrich and M. B. Mineev-Weinstein (1997). ROAMing terrain: Real-time Optimally Adapting Meshes, Proceedings. Visualization '97, pp. 81-88.

[18] Han, G.; Liu, L.; Chan, S.; Yu, R.; Yang, Y. HySense (2017). A Hybrid Mobile CrowdSensing Framework for Sensing Opportunities Compensation under Dynamic Coverage Constraint. IEEE Commun. Mag., 55: 93–99.

[19] Pornsing, C.; Sodhi, M.S.; Lamond, B.F. (2016). Novel self-adaptive particle swarm optimization methods. Soft Comput., 20: 3579–3593.

[20] Karaboga, D. & Gorkemli, B. (2014). A quick artificial bee colony (qABC) algorithm and its performance on optimization problems. Appl. Soft Comput., 23: 227–238.

[21] Yang, X. S. (2012). Flower Pollination Algorithm for Global Optimization. In Proceedings of the 11th International Conference on Unconventional Computation and Natural Computation, Orléans, France, 3–7 September; Springer: Berlin/Heidelberg, Germany, 2012; pp. 240–249.

[22] Wu, X.; Shao, H.; Wang, S.; Wang, W. (2019). A Multi-objective Artificial Flora Optimization Algorithm. In: Song H., Jiang D. (eds) Simulation Tools and Techniques. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, vol 295.

[23] Alsweis, M. & Deussen, O. (2006). Wang-Tiles for the Simulation and Visualization of Plant Competition. Computer Graphics International, pp. 1-11.