Optimizing Boids Algorithm using Density Based Clustering Algorithms and Characterizing Emergent Behaviour using HDBSCAN

Rishabh Narayan

Graduate Student-M.Eng in ECE

Department of Electronics and Computer Engineering,

University of Waterloo, 200 University Ave W, Waterloo, ON N2L 3G1

Abstract -

This paper describes an implementation of the Boids algorithm by simulating a flock behaviour in artificial agents (boids). The boids algorithm's original implementation however has an asymptotic complexity of $O(n^2)$, which is not an issue when working with smaller flock sizes, but degrades the algorithm's performance for increasing flock sizes. The clustering algorithms, DBSCAN and HDBSCAN are implemented in a Python environment and provide us with an improved performance of the boids algorithm. The performance is estimated using the average FPS of our simulations over a specified time period. While the DBSCAN algorithm gives a slight improvement over the original boids algorithm, the HDBSCAN algorithm achieves a drastically better performance than the original boids algorithm. The results of the simulations are statistically verified by conducting a one-way ANOVA test to detect any significant difference between the algorithms and a post-hoc test is further conducted to compare the effect between the original algorithm and the two clustering algorithms.

The use of HDBSCAN algorithm for simulating the flock behaviour in the boids, gives rise to a novel emergent behaviour in the flock simulations. This new emergent behaviour is characterized and observed to understand its nature of occurrence.

Index Terms – Boids algorithm, DBSCAN, HDBSCAN, ANOVA and post-hoc test, emergent behavior.

I. THEORETICAL BACKGROUND

A. Boids Algorithm

The boids algorithm was developed by Craig Reynolds in 1986 [1], the goal of the algorithm was to simulate the flocking behaviour of birds amongst artificial agents called "bird-oid objects"-boids. The phenomenon of emergent behaviour is observed in the boids simulation as a result of complex yet organized group (flocking) behaviour arising from simple interaction rules between the generated boids [1].

The boids algorithm is one of the prime examples of simulating emergent behaviour. The algorithm simulates complex flocking behaviour not by controlling the behaviour of the entire flock but instead by only controlling the behaviour of each individual boid agent in the simulation. Each boid is governed by three primary heuristic rules – Cohesion, Alignment and Separation [2]. The cohesion rule states that the individual boid will move towards the centre of a nearby flock of boids, by calculating the average position of the flock. This ensures that none of the boids drift astray from the global flock. The alignment rule is responsible for orienting the boids' direction. This rule causes the boids of a given flock to orient themselves towards the average direction of the flock. By implementing this rule, we make sure that the boids don't move away in different directions and maintain the orientation set by its neighbouring flock of boids.The final rule is the rule of separation. This is a crucial rule to implement as it ensures none of our boids collide with one another when exhibiting flock behaviour. The boids are thus programmed to steer away if they come too close to their neighbouring boids.

The algorithm has found to be of great use as a framework for animators and graphic designers to use for simulating flock behaviour in animations or films. Further this behaviour is of great use in simulating and predicting crowd behaviour [3].

B. Need for optimization of Boids Algorithm

The original implementation of the boids algorithm has an asymptomatic complexity of $O(n^2)$ [2] and hence simulating large flocks of boids or simulating flock behaviour in a boid dense environment is computationally expensive. In the boids model, the boids are governed by the aforementioned rules of cohesion, alignment and separation, but for each boid

to implement any of these rules it needs to first determine the positions of its neighbouring boids.To do so, each boid in the original implementation of the algorithm has to consider the position of all the boids in the flock to determine which boids are not its neighbour and which boids are. This manner of determining neighbours, makes it hard to simulate boid algorithm for large flocks and denser flocks in real-time systems. It is hence important to resolve this by exploring optimization techniques for reducing the complexity of the original implementation [4].

Use of spatial data structures and spatial hashing algorithm has showed an improved performance in the boids simulation [2][7]. Clustering algorithms, especially DBSCAN have also shown improvement in the performance of the simulation [5]. In this paper, DBSCAN and HDBSCAN are implemented for optimizing our boid's performance.

C. DBSCAN Algorithm

The DBSCAN (Density-based spatial clustering of applications with noise) algorithm falls under the category of unsupervised machine learning models.

DBSCAN essentially clusters points together which are close to one another, the closeness being determined by a distance metric and the number of points to clustered being decided by a minimum number of points. The DBSCAN algorithm thus requires two primary parameters for its implementation. The parameters being eps (epsilon) and minPoints. The eps essentially gives a specification of how close the points within a cluster have to be, to be considered a cluster. The minPoints parameter sets the minimum number of points required for a cluster to form [6].

Many previous implementations of the DBSCAN algorithm for the boids simulation have yielded an improved and optimized output, hence this clustering algorithm was selected as an optimization method [5] [7].

D. HDBSCAN Algorithm

HDBSCAN (Hierarchical density-based spatial clustering of applications with noise) is an extension of the original DBSCAN algorithm which accounts for varying densities of clusters and not a fixed one as was the case with DBSCAN. In addition to being better for data with varying density, it's also faster than regular DBSCAN [8].

The algorithm of HDBSCAN essentially starts of the same as the original DBSCAN, by estimating the core distance of every point so as to know of its neighbours for the given number of minimum samples, defined by min_samples. Next, the potential clusters which can be formed, form a dendogram but unlike in DBSCAN, in HDBSCAN the dendogram is not cut off based on a set epsilon but instead the algorithm adds hierarchy to the clusters and the clusters with the highest density end being selected in the end. In this manner, the HDBSCAN algorithm traditionally performs better than DBSCAN and provides clusters of higher densities [9]. No precise study exists on the use of HDBSCAN for implementing the boids algorithm, it is chosen for this study as it has yielded better results for clustering problems than DBSCAN [10].

II. HARDWARE AND SOFTWARE CONFIGURATIONS

For this experiment, a PC with AMD Ryzen 7 5800H @3.20 GHz and 8GB RAM were used to implement the algorithms. The code for the same was run in a Python 3 environment using the PyCharm IDE and the statistical verification was carried out in the language R using the RStudio IDE.

III. METHODOLOGY

A) Boids Algorithm Implementation

The boids algorithm is implemented in a Python 3 environment and the individual behaviours of the boids are programmed according to the aforementioned rules of cohesion, separation and alignment.

In the file Boid_Behaviour.py, the behaviour of the boids is defined. The code structure for the boid_behaviour.py was referenced from a github repository [11].We initially create a Boid class, this class consists of the various behaviours and properties of the boid. We store the properties of position update, velocity update, colour and configurations within our class. The position update function is defined so as to update the boid's position with respect to its environment every time it updates its velocity.

The three rules of cohesion, alignment and separation are implemented in the functions centring_flk, velMatching and avoid_coll, respectively. In the centring_flk function, which is

for the cohesion rule, we make sure that the boids steer towards the center of mass of local flockmates. The boids in our simulation are all of the same mass and hence their centre of mass will be equivalent to their average position. We hence calculate the centre of mass for all the boids and then subtract the boid's current position from it. This binds the cohesion behaviour in to our boids. The above expression is calculated like so,

[xTotal / num - self._posn[0], yTotal / num - self._posn[1]]

As depicted in the formula above, we are simply returning the x and y coordinates of our boids after subtracting their position from the centre of mass of all the boids. In the velMatching function in our code, we implement the alignment rule of the boids. The alignment rule ensures that the boids to orient themselves towards the average direction of the flock. The average direction is calculated from the velocity vector of our boids which we instantiate in our Boid class as _vel. The velocity of the boids is then subtracted from this average direction to give the velocity vector with our new aligned direction. This above expression is calculated like so in our function velMatching,

[xTotal / num - self._vel[0], yTotal / num - self._vel[1]]

Our final rule of separation is implemented in the avoid_coll function. The rule of separation ensures that our boids do not overcrowd within a given area or crash into one another. We then create a new python script algorithms.py wherein a class World is created. Upon instantiating the class object our variables get initialized. The variables within our class World include the width and height of the boids, the number of boids, the configuration of boids and other variables associated with our clustering algorithms which we will discuss later. We define a function, boidlocalupdate which we call every time our boid's update is scheduled. In the simulation.py file, we define our update function which is responsible for updating the boid's velocity and positions. This update is scheduled using the library pyglet's function called, pyglet.clock.schedule().This schedule() helps run our simulations at the highest possible frequency, by calling the update function as frequently as possible and uses the most RAM in a CPU [12]. We do this since we want to test the performance of our algorithms when they are required to produce their maximum efficiency.

B) DBSCAN Algorithm Implementation

DBSCAN is an unsupervised learning clustering algorithm which produces clusters based on the density of points. Using DBSCAN for the boid algorithm is a popular technique of optimizing the boids algorithm [5][7]. Here we end up creating small clusters of boids, i.e., sub flocks within which the boids only consider their neighbouring flock mates for its velocity and position update, unlike how in the original algorithm the boids had to consider all the boids in the flock. The sub-flocks created by clustering are usually far apart from one another and don't interact that often. The DBSCAN implementation is performed in our algorithms.py file. The sklearn.cluster [13] library is imported and the DBSCAN function is applied within our aforementioned function of boidlocalupdate(). The DBSCAN function has two parameters in eps and min_samples. We assign eps to be equal to our boidRange which is the range for which we want our boids to localize within. The labels are extracted using the _labels method and we set the number of our clusters to the length of the labels array. Then these clusters are called every time our boids update. This implies that when fitting our DBSCAN model on our boids positions the boids only consider their cluster's flockmates while updating its position. This is the primary difference between the original algorithm and the clustering approach that produces clusters based on the density of the points.

C) HDBSCAN Algorithm Implementation

The HDBSCAN library [8] is not a part of any mathematical python library and has to be installed using pip install or conda install*. The library is imported as hdbscan in our algorithms.py script and is implemented under the boidlocalupdate() function just like our DBSCAN algorithm as well. The HDBSCAN does not require an epsilon to be defined as an input parameter [8], however in our case we need to provide the boids a specified range within which it is to search for its nearest neighbours. The min_cluster_size parameter when chosen with a small value can result in many micro clusters. These need to be merged using the epsilon value in form of the cluster_selection_epsilon parameter in the hdbscan model. It essentially makes sure that no further split ups occur in clusters below the given epsilon value. This in fact improves the clustering performance of our model drastically [8]. The hdbscan is applied to our boids algorithm in a similar way as our previous dbscan algorithm. The epsilon value is set equal to the boidRange

which determines at what threshold level our clusters stop splitting up further.

D) Parameter Setup

We set the following parameters given in table 1, for all the simulations of our three algorithms, i.e., the original boids algorithm, DBSCAN algorithm and HDBSCAN algorithm. In order to avoid any excess increase or decrease in speed, we restrict the speed between boidspd_max and boidspd_min, analogous to the parameters taken by Maruyama et al [5]. The weight values associated with our three rules of cohesion, alignment and separation are set in such a way to give way more weightage to the separation parameter so that our boids don't overcrowd in our environment. The parameter values of size and range are selected to observe larger flocks.

Parameter	Value
boidRange	1600 pixel^2
boidcoll_weight	4
boidvel_weight	0.5
boidcentring_flkWeight	0.3
boidspd_max	60
boidspd_min	45
boidSize	6

Table 1. Parameter Values

E) Performance Estimation and Data Collection

The performance of our three algorithms, i.e., the original boids algorithm, the DBSCAN algorithm and the HDBSCAN algorithm is estimated using the FPS (frames per second) of our final simulations. FPS is a common unit of measurement when measuring boids algorithm's performance [14]. The FPS essentially computes the average computation time for all boids per frame, implying that an increase in the FPS of our simulations is directly correlated with an increase in the efficiency of calculation of the boids new updated positions. Setting the parameters as mentioned earlier, we compare our three algorithms' FPS with the number of boids in our simulation environment and the area of the simulation environment. The O(n2) complexity of the original boids algorithm [2] implies that with an increase in the number of boids and with an increase in the density of boids in the environment area, the FPS of our simulation for the original algorithm should drop significantly. To optimize this very complexity, we use our two clustering algorithms of DBSCAN and HDBSCAN. Now in our simulation.py file we define a function to call the pyglet library's method pyglet.clock.get_fps(). This provides us with the FPS of our simulation. Now by using the pyglet method of pyglet.clock.schedule_interval(), we call for the FPS of our simulations to be collected every second and stored in an empty array (fps_arr). We then run our simulation for a period of 60secs and take the average of the FPS across our runtime. Hence upon running our simulation, we get an output of the average FPS over a 60sec runtime of our simulation. We then note the values of our FPS for all three of our algorithms while varying the number of boids to study the boids v/s FPS relationship and the area of the environment to study the area v/s FPS relationship. When noting down the values of FPS while varying the number of boids, the environment area is kept constant at 230400 pixel^2 (which is set by keeping the World dimensions in the simulation.py file at 640,360). Similarly, while varying the area of the environment for obtaining FPS values, we keep the number of boids constant at a value of 200. The observed values are then stored in two CSV files, which are then used to visualize our results and statistically verify our results.

F) Statistical Tests

A one-way between subject ANOVA test is conducted to detect if there is a statistical difference between the algorithms when measuring FPS over number of boids or over area of environment. Further a post-hoc test (Tukey's test) is conducted to compare the effect between the three algorithm groups. The significance level is set at α=0.05 level for these tests. The orginal CSV files with our data observations are formatted by factoring the three algorithms as – 1 – "original boids algorithm", 2 – "DBSCAN algorithm" and 3 – "HDBSCAN algorithm". So, the modified CSV files consist of two main columns of FPS readings and that of Algorithm, which consists of the factorized values of our three algorithms.

The CSV files are imported in our Rmd file on the RStudio IDE. The car library is imported in our R environment and its ANOVA package is implemented on our datasets of boids v/s FPS and area v/s FPS. The results are summarized and subsequently a Tukey's test is conducted for determining the statistical effect between our three algorithm groups.

G) Visualization

The simulation of our three boids algorithms is carried out using the pyglet library in python. The pyglet library is imported in our simulation.py file

and the draw function for the boids is defined. The algorithm to be simulated can be selected by giving the variable rangeclust the respective algorithm's categorical value, i.e., 0 = "original algorithm", 1 = "DBSCAN" and 2 = "HDBSCAN". Then, upon running the pyglet.app.run() command, our simulation for the selected algorithm is implemented and we get the average FPS value of our simulation every 60secs. The data collected from our simulation runs is stored in two CSV files, one for the data for varying number of boids with our FPS and one with data for varying the area of our boid environment with our FPS. Using the matplotlib library in python, we visualize the results of our observed data in our VizBoids.ipynb file. Further visualization of our Tukey's post-hoc test is carried out in RStudio itself.

IV. RESULTS AND ANALYSIS

In this study, two clustering algorithms, i.e., DBSCAN and HDBSCAN were implemented for optimizing the original boids algorithm. The hypothesis of this study was to establish that one of the clustering algorithms optimizes our original boids algorithm. To compare the performances our algorithms, we measured the FPS of the simulations and varied the number of boids in an environment and the area of the environment against the FPS of our simulations. Thus, data has been collected in two files, one being for the observations of number of boids v/s FPS and one being for the observations of area of environment v/s FPS.

The data points are then plotted for our two categories.

A) FPS v/s Number of Boids

The observations of our FPS and Number of boids are plotted across one another with our three algorithms as the labels. The graphical plot is shown in fig 1.

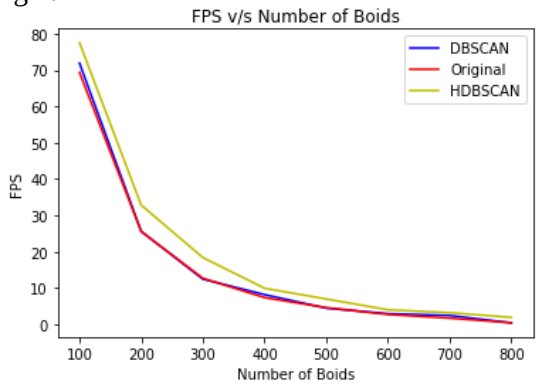

Fig 1. FPS v/s Number of Boids

The original boids algorithm is known to degrade in performance with an increase in the number of boids in the simulation environment owing to its O(n2) complexity [2]. We find this to be true as we can see the original algorithm's FPS drops drastically upon simulating for 200 boids. The surprising observation however is that our DBSCAN algorithm shows no improvement over our original boids algorithm. The DBSCAN's FPS experiences an exact similar drop off as that of the original algorithm when simulating for 200 boids. The HDBSCAN shows a marginally better performance than the other two algorithms and has a higher average FPS than the other two algorithms for up till 400 boids. It then exhibits a similar behaviour as the other two algorithms. We don't notice any significant improvement over the original boids algorithm as despite optimizing the boid algorithm with the clustering algorithms, we still observe the O(n2) complexity arise when the number of boids in our given environment is increased.

B) FPS v/s Area of Environment

The observations of our FPS and the area of the environment are plotted across one another with our three algorithms as the labels. The graphical plot is shown in fig 2.

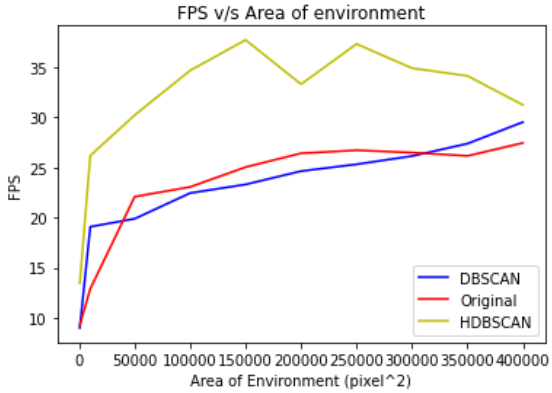

Fig 2. FPS v/s Area of Environment

The increase in area of the environment yields a better performance owing to the decrease in the boid density of the environment [2]. The original algorithm is observed to have an FPS in the range of 10 to 25 and even performs better than the DBSCAN algorithm for areas between 50000 and 300000 pixel^2. The DBSCAN again yields a relatively poor performance for the majority of the areas but shows a slight improvement when

approaching areas beyond 350000 pixel^2. The HDBSCAN on the other hand performs remarkably with a significantly higher FPS than the other two algorithms for all the values of area. The FPS of the HDBSCAN reaches quite high values in the range of 100000 to 350000 pixel^2. The HDBSCAN algorithm provides us with satisfactory results in this simulation and intuitively proves our hypothesis of a clustering algorithm optimizing the boids algorithm.

C) Statistical Verification

The above observations from the Fig 1 and Fig 2 give us an intuitive understanding of the algorithms performing poorly for an increase in the number of boids, while HDBSCAN performs significantly better than the original algorithm and the DBSCAN algorithm when area of the boid environment is increased. Using our formatted CSV files and performing ANOVA and post-hoc Tukey's test on our datasets, we statistically verify our intuitive understanding of the three algorithms' performance.

i) FPS v/s Number of Boids

We conduct a one-way between subject ANOVA on the algorithm type and the FPS. The main effect on algorithm type was not found to be statistically significant (F=0.058; P=0.944). The p-value is observed to be 0.944 and is greater than our significance level of 0.05, thus we cannot reject our null hypothesis of there being no statistical significance. This signifies that none of the algorithms perform well enough with an increase in the number of boids. Further a Tukey's post-hoc test is conducted to compare the effect between the three algorithm groups. Using Tukey's HSD analysis and on observing from the Tukey's plots given below in Fig 3 it is found that the algorithm types are not significantly different from one another as (p>0.05) for all cases.

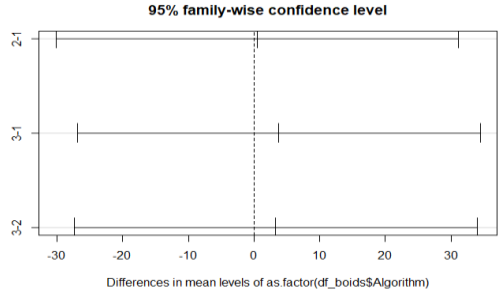

Fig 3. Tukey's Plot for Number of Boids

The 1,2,3 on the Y-axis refer to our three boid algorithms (1-'original algorithm', 2-'DBSCAN' and 3-'HDBSCAN'). There is no significant difference in the mean levels of all three algorithms as all their interaction terms have a p-value (0.99, 0.94, 0.96) > 0.05. We can notice the slight deviation of the plot for 3-1 and 3-2 from 0, which explains the slight improved performance of HDBSCAN (3) over the other two algorithms (1 & 2). However, this statistically verifies that the three algorithms performed poorly with an increase in the number of boids.

ii) FPS v/s Area of Environment

We conduct a one-way between subject ANOVA on the algorithm type and the FPS. The main effect on algorithm type was found to be statistically significant (F=6.085; P=0.00658). The p-value is observed to be 0.00658 and is less than our significance level of 0.05, thus we can reject our null hypothesis of there being no statistical significance. This implies that our hypothesis of an algorithm optimizing the original boids algorithm is true. Further a Tukey's post-hoc test is conducted to compare the effect between the three algorithm groups. Using Tukey's HSD analysis and on observing from the Tukey's plots given below in Fig 4 it is found that the algorithm types are significantly different from one another as (p<0.05) for all cases.

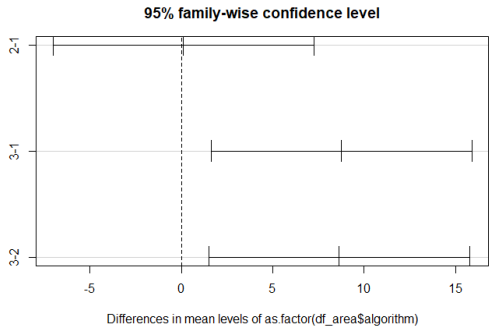

Fig 4. Tukey's Plot for Area

There is a significant difference in the mean levels of two interaction terms, i.e., 3-1 and 3-2. Here 3 is the HDBSCAN algorithm and 1 and 2 refer to the original and DBSCAN algorithms respectively. The p-value of the 3-1 interaction term is 0.013(<0.05) and the p-value of the 3-2 interaction term is 0.015(<0.05). There is however no significant mean difference between the interaction term 1-2 as its p-

value is 0.99(>0.05). This signifies the drastically improved performance of our HDBSCAN algorithm over the other two algorithms and also shows that the DBSCAN algorithm does not optimize the original boids algorithm. This statistically verifies that the HDBSCAN algorithm does provide a better performance as compared to the original algorithm for a growing area of the boid environment.

D) Visualizations

(i) The Original Algorithm

The original boids algorithm displays a global flock behaviour as shown in Fig 5 below.

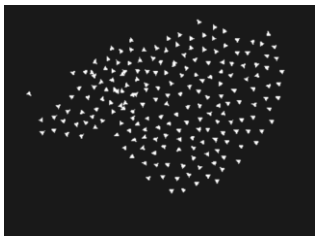

Fig 5. Original boids algorithm flock display

(ii) DBSCAN Algorithm

The DBSCAN algorithm displays a clustering behaviour while flocking and these mini sub flocks can be noticed due to their differing colours in Fig 6.

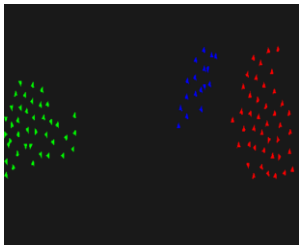

Fig 6 DBSCAN boids – flock display

(iii) Novel Emergent behaviour using HDBSCAN

The HDBSCAN implementation displays flocking behaviour just like the other two algorithms. However, whenever two or more clusters of flocks approach towards one another, the boids seem to scatter instantly almost as if disoriented and then immediately form back into their flocks again. This sort of "disoriented scattering behaviour" can be thought to be analogous to the sort of behaviour birds exhibit upon rare clash of flocks. These instances though rare, have been recorded quite recently when a flock of geese clashed into a flock of starlings. The birds seem to scatter around as if disoriented before returning to their flocks again [15].

This disoriented scattering behaviour is shown below in Fig 7.

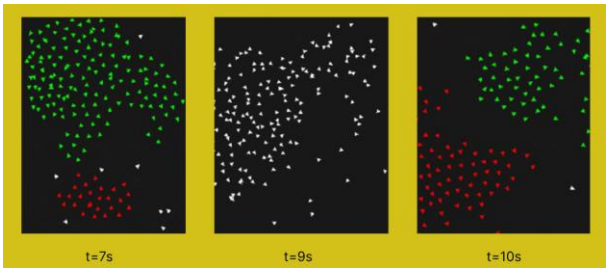

Fig 7. Step by step display of "disoriented-scattering behaviour"

We can observe that at time step t=7secs two clusters of different density approach one another. At t=9secs the boids of both the flocks lose their cluster identity (i.e., disorientation) and then they scatter around haphazardly before immediately reflocking back into their clusters at t=10secs. This behaviour is unique and shows how "the interrelations between the entities making up the system are not permanently established but evolve along time." [16]

V. CRTITICAL EVALUATION AND FUTURE WORK

We introduce the boids algorithm in this study and then due to its O(n2) complexity, try to optimize it using clustering-based algorithms. In related studies where the DBSCAN algorithm is implemented for boid optimization, it has mostly yielded a statistically significant performance [5][7]. However, in our implementation the DBSCAN fails to optimize our original boids algorithm with an increase in the number of boids and with the increase in the area of the environment. The DBSCAN implementation requires slight optimization and can be combined with other spatial hashing algorithms such as tiling to yield better results. The HDBSCAN's performance with respect to the number of boids is another limitation of this study. Further due to time constraint the characterization and in-depth study of the emergent behaviour was not possible. This "disoriented-scattering" behaviour can further be studied and taken up for future work. The interaction network

can be analysed to determine the global properties of the graph, as proposed by Moncion et al[16] when characterizing emergent behaviour in complex systems [16].

VI. CONCLUSION

This study's objective was to optimize the boids algorithm using the clustering algorithms. While the DBSCAN clustering algorithm failed to yield satisfactory results, the HDBSCAN clustering algorithm yielded highly statistically significant results. The HDBSCAN achieved quite high FPS values when varied with the area of the boid environment. It outperformed the original boids algorithm and the DBSCAN algorithm by a significant margin. One-way ANOVA and post-hoc Tukey's test further validated this study's findings. Further, novel emergent behaviour using the HDBSCAN implementation was characterised. The analysis of its interaction network is left for future work.

VII. REFERENCES

[1] Craig W. Reynolds. 1987. Flocks, herds and schools: A distributed behavioral model. SIGGRAPH Comput. Graph. 21, 4 (July 1987), 25–34. DOI:https://doi.org/10.1145/37402.37406

[2] Boids by Craig Reynolds, https://www.red3d.com/cwr/boids/

[3] Chiang, Ching-Shoei & Hoffmann, Christoph & Mittal, Sagar. (2013). Emergent Crowd Behavior. Computer-Aided Design and Applications. 6. 865-875. 10.3722/cadaps.2009.865-875.

[4] Mavhemwa, Prudence M. and Ignatius Nyangani. "Uniform spatial subdivision to improve Boids Algorithm in a gaming environment." International Journal for Advance Research and Development 3 (2018): 49-57.

[5] Maruyama, Norihiro & Saito, Daichi & Hashimoto, Yasuhiro & Ikegami, Takashi. (2019). Dynamic organization of flocking behaviors in a large-scale boids model. Journal of Computational Social Science. 2. 10.1007/s42001-019-00037-9.

[6]scikitlearn DBSCAN library -https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html

[7] Optimising boids algorithm with unsupervised learning by Adam Price https://towardsdatascience.com/optimising-boids-algorithm-with-unsupervised-learning-ba464891bdba

[8]https://hdbscan.readthedocs.io/en/latest/how_hdbscan_works.html

[9] Lightening Talk : Clustering with HDBSCAN by Brendan Bailey - https://towardsdatascience.com/lightning-talk-clustering-with-hdbscan-d47b83d1b03a

[10] Strobl, Michael & Sander, Joerg & Campello, Ricardo & Zaïane, Osmar. (2021). Model-Based Clustering with HDBSCAN*. 10.1007/978-3-030-67661-2_22.

[11]https://github.com/adamprice97/Boids/blob/master/Boid.py

[12]https://pyglet.readthedocs.io/en/latest/programming_guide/time.html

[13]https://scikit-learn.org/stable/modules/classes.html#module-sklearn.cluster.

[14] Lindqvist, S. (2018). Performance Evaluation of Boids on the GPU and CPU (Dissertation). Retrievedfrom http://urn.kb.se/resolve?urn=urn:nbn:se:bth-15970

[15]https://www.mensjournal.com/adventure/flock-geese-acts-like-bowling-ball-colliding-another-bird-flock-video/

[16] Thomas Moncion, Patrick Amar, Guillaume Hutzler. Automatic characterization of emergent phenomena in complex systems. Journal of Biological Physics and Chemistry, Basel, Switzerland: Collegium Basilea (Institute of Advanced Study); Tbilisi: Association of Modern Scientific Investigation (AMSI), 2010, 10, pp.16–23. hal-00644627