Hybrid Approaches of Swarm Intelligence and Genetic Algorithm

Eric Siqueira de Carvalho
Institute of Computer Science
University of Göttingen
Göttingen, Germany
e.siqueiradecarval@stud.uni-goettingen.de

Surendhar Muthukumar
Institute of Computer Science
University of Göttingen
Göttingen, Germany
surendhar.m@stud.uni-goettingen.de

***Abstract*—Swarm algorithms are highly used for optimization problems, and have received increasing attention due to its potentials. Genetic Algorithms and Particle Swarm Optimization are a few examples of meta heuristics methods, and although they share many characteristics, different problems take advantage of each one. In order to make use of the best of both algorithms, hybrid approaches have been developed. This paper explores these hybrid approaches in depth and perform a practical experiment to better understand these approaches. Furthermore we explore the implementation of genetic algorithm to improve the clustering of images based on ant's odors [1]. We propose one hybrid approach that fine tune the hyperparameters of AntClust, which was able to slightly improve the test results. We also propose a method to apply genetic algorithm to the rule set based on the original article [2], and although it did not succeed due to the complexity of a rule set, it could serve as the base for further work.**

***Index Terms*—Swarm Algorithms, Particle Swarm Optimization, Evolutionary Algorithms, Genetic Algorithm, Hybrid Approaches, AntClust, Swarm Intelligence**

I. Introduction

Immense amount of data is generated in this technological era through various medium such as social media, scientific experiment, telecommunications and so on. Structured analysis performed on generated/available data yields valuable insights in any field of study, this extraction of insights is called the Knowledge and Data Mining (KDD). Intense analytical algorithms like pattern discovery, classification, machine learning are used in KDD [3]. Optimization based pattern discovery is one key technique used here [4]. Clustering and classification are processes that are performed on the data to convert it into machine understandable formats in lexical contexts. The efficiency of such processes are determined by the underlying algorithms. Swarm intelligence (SI) has been observed as an efficient algorithm with high degree of generalization [4]. In clustering, the center of a cluster, called the centroid, changes during every iteration based on the dataset distribution. Multiple clustering approaches are used based on the problem space, where the two most popular are hierarchical and partition-based approaches. Partition, as name suggests, divides data based on the similarity and dissimilarity between the clusters. Finding similarities and dissimilarities are carried out by sub-processes. Hierarchical approach forms a hierarchical tree of classification, with condition that an element of the dataset does not shift to another cluster during the process. Both the approaches have some drawbacks due to the dependencies on defining the cluster numbers initially and also the performance difference for different shape of data. To counter the drawbacks, optimization technique is coupled to the clustering to get better results. Swarm intelligence is one such optimization technique to increase performance of the clustering. The underlying principle of SI is to perform the optimization based on the approach of mimicking the behaviour of naturally available swarm communications of organisms.

Swarm Intelligence as the name suggests is inspired by the collective behavior of organisms that exists as pack or group and perform organized activities with minimal interpersonal interactions [5][4]. The Swarms are decentralised, self organized with reduced internal communications. Primary applications of these SI algorithms are in the fields of Optimizations as these algorithms have the tendency to handle multi-variable optimization tasks. Some of the examples of the application of SI involves data mining, network managements, minimization of traversal distances in terms of logistics and so on. The SI techniques are based on collective behaviour exhibited by insects and animals. Ant, bee, wasp, and firefly contribute towards the insect based SI algorithms while lion, wolf, and monkey contribute towards the animal-based algorithms. Initially SI algorithms and techniques were used for traditional optimization problems but inclining trends have been noticed in other areas of research as well like KDD, as well as complex NP-hard problems. Ant colony optimization and particle swarm optimization are two major widely used SI techniques for solving discrete and continuous optimization problems.

This work discussed the significance of particle swarm optimization (PS0) and genetic algorithms (GA) with a practical implementation of the same in a problem solving scenario. Additionally the possibility and effectiveness of combining these 2 techniques is also briefly discussed. In section II the PSO methodology and its application is discussed, while section III does the same for GA. In section IV both methods previously described are compared. Many state of art works using hybrid approaches are addressed in section V. To further explore the value of hybrid approaches, section VI simulate and compare standalone approaches with hybrid ones for a classic optimization problem. In section VII we apply a hy-

bridization method into the proposed application of AntClust [1]. Finally in section VIII we summarize our findings and discuss further work directions.

II. Particle Swarm Optimization

Particle swarm optimization is meta-heuristic algorithm initially developed by mimicking the collective behavior of swarm of birds. The intelligence observed from this swarm behavior allows to replicate complex global patterns irrespective of the solution space. In PSO individuals of a swarm are termed as particles and such particles are members of a highly decentralized self-governed intelligent environment. Particles follow very simple rules to contribute to the environment with minimal intra particle communication, aiding to a complex global behaviour. With minimal individual effort, a global complex behavior is established and hence the optimization process is carried out through the exploitation of the formed global behavior. Three main components of PSO are particles, social & cognitive components of particles, velocity of particles [4].

General optimization problems have huge solution space with multiple feasible solutions. Depending upon the nature of the problem space there might be multiple local maxima/minima, the global solution is efficiently obtained through optimization. In PSO, the particles represent individual solutions. This optimization process moves towards achieving the optimal solution through iterative particle movement in swarm, guided by 2 scores, social learning (pBest), cognitive learning (gBest). The pBest value is the best of the particle solution and the gBest value is the combined learning of the swarm obtained through the best position the swarm achieved. Swarm guides the particles' movement during iterations and the velocity of the movement is determined by the scores pBest & gBest. Iterations continue till the swarm attains the global solution.

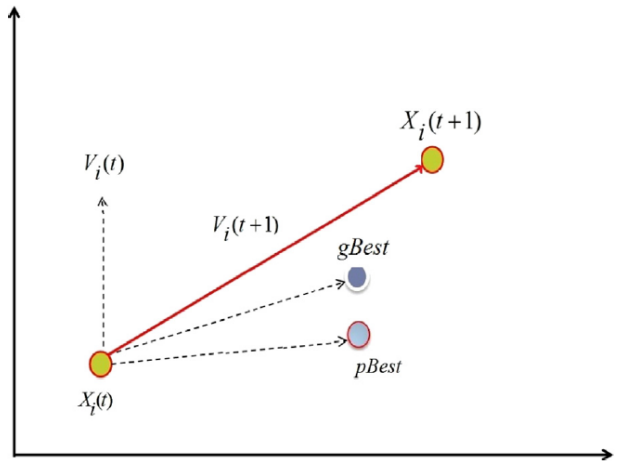

Figure 1: Particle movement in PSO [4].

Fig 1 shows the movement of a particle X_i between two iterations t and t+1. The particle moves from a position $X_i(t)$ to $X_i(t+1)$, influenced by 3 core parameters of the PSO pBest, gBest, $V_i(t)$. pBest gives the direction of personal best fitness score of the particle while gBest gives direction of the global best fitness score, $V_i(t)$ is the direction of current velocity of particular particle. Particle moves along the direction of the 3 vectors (direction parameters) and reaches the new position. Now the algorithm checks the fitness score of the particle using the objective function and updates the core parameters accordingly. There are chances that pBest and gBest change to different value than before while velocity direction would change to $V_i(t+1)$ as shown in the figure. This process happens to all the particles of the swarm in every iteration until convergence.

Typically for an optimization problem, the swarm parameters (number of particles, pBest, gBest, Velocity of particles) are initialized randomly for the start of the iterative process. The number of particles to be initialized differ based on the undertaken problem, generally initial particles count is 20 - 40 [4]. The random particle initialization affects the speed at which the optimum solution is reached, but this could be handled through hybrid approaches for the initialization of the swarm parameters. Algorithmically PSO-based optimization involves objective function for the swarm to optimize, the movement mechanism that governs the swarm movement. Multiple objective functions can also be used in case of complex problems. PSO has wide range of applications in multiple fields like Health-care, Industry, Commercial establishments, Smart city projects, Environmental analysis and so on. So standard application topics would be resource/demand estimation, detection & classification of item of focus, segmentation, scheduling, resource optimization, risk assessment. PSO exhibits good efficiency in solving above problems, while studies are still being carried on to identify various other use cases to expand the use of PSO.

III. Genetic Algorithms

Genetic algorithm (GA) is a search method to find exact or approximate solution to an optimization problem, the method is also extended for search problems. GA follows adaptive search heuristics falling under the category of Evolutionary Algorithm. This technique follows the principles of evolutionary biology like mutation during inheritance, natural genome selection and genetic cross overs [6]. Similar to the biological evolution, the GA algorithm performs automatic improvement of the individuals through natural selection and genetics evaluation. GA starts with set of solutions called population from the solution space of the problem, the initial population is achieved through randomized selection of a set of the solutions to start the algorithm. Every solution in the population is represented as chromosome to comply with the underlying biological evolutionary architecture. This is iterative optimization process where every iteration is termed a generation. Fig 2 represents the flow of process performed during an iteration of the GA technique. During every generation a pool of fit chromosomes of the current population is selected based on the fitness scores using the underlying objective function(s).

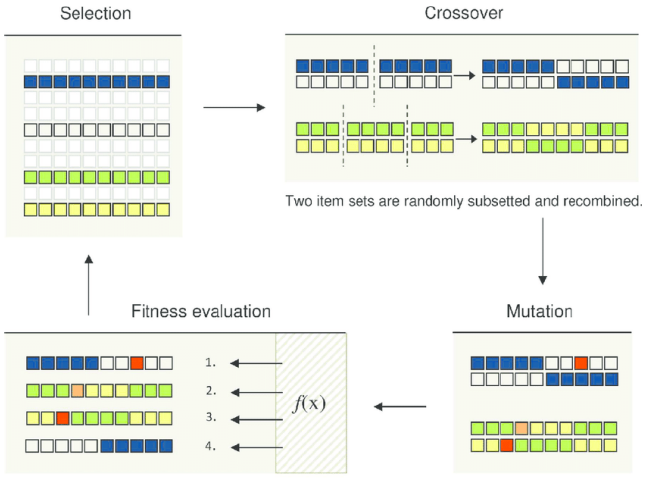

Figure 2: Genetic Algorithm iterative flow diagram [7].

The selected fit chromosomes are allowed to produce off-springs through random mating, cross overs, mutations. Intrinsic probabilistic scores of selected chromosomes in the pool allow interbreeding, mutations to produce off-springs, the new chromosomes produced as a result of the evolutionary process contribute to the population, conserving the population strength through out the generations. End conditions are used to terminate the evolutionary iterations. General end conditions are as follow:

- Algorithm finds solution that satisfies minimum criteria defined.
- Certain defined number of generations are performed.
- Limits on cost factor of the computation.
- Resource allocation limitation.
- Reaching plateau condition for fitness change during successive generations [6].

GA differs from general search algorithm by using evolutionary probabilistic strategy, simulating the natural selection and working with the coding of the parameter set rather than the actual parameters. Problem parameters are represented as binary string, the length of the binary string consisting 1 and 0 directly imply with the probability of the individual chromosomes on the solution search [8]. Additionally coding the parameters enables the solution space to be in numerical paradigms aiding to the computations involved int the algorithm. GA is better suited for solving optimization problems on scheduling, shortest path searches, as the technique simultaneously tries multiple possibilities on exploring the solution space and evolves the best among the population.

IV. Comparison

Both Particle Swarm Optimization and Genetic Algorithm Optimization follow similar steps as they are inherited techniques from Evolutionary Algorithm paradigm. But there are certain aspects of these mentioned algorithms which make them differ from one another. Such key aspects are mentioned below:

A. Generation of Initial population

GA and PSO involves random initialization of the starting population to begin the algorithm. While the population among the solution space is gathered in random manner, certain parameters of the algorithm are calculated on assumptions applicable to the nature of the problem. This random initialization might lead to delayed convergence of the technique to optimal solution. Hybrid approaches on clustering have been used in [9]. In this hybrid approach K-means is used to obtain the initial clusters than using some random centroids, further optimization of the clusters is performed by PSO. Thus multiple hybrid approaches are possible for the initialization of the swarm in PSO.

B. Time Complexity

Both the algorithms aim to find the optimum solution in the solution space using different underlying approach. The principle is to search the solution space and exploit it as much as possible in search of optimum. The extent of exploitation can be increased by increased the participating individuals number (population count). The trade-off between better exploration and faster convergence is different among GA and PSO. In GA, sorting of fitness scores of the involved chromosomes is done during every iteration. An exponential increase in time complexity is observed while increasing the population strength, hence increasing the resource and time required for the algorithm to convergence to optimal solution while using increased population strength. As PSO does not sort the particles based on the score it maintains linearity in the time complexity-swarm strength relation [10] [7]. Instead of sorting, the algorithm only selects the best among the fitness scores achieved by the particle in a lifetime basis, which only needs simple simple list/array operations.

C. Early Convergence

PSO exhibits high affinity towards moving in the direction of the gBest which improves the tendency to cluster rapidly and the swarm might reach a stagnant condition. Possibilities for observing plateau in the best fitness score trend are common in PSO, indicating the early convergence. GA is free from dominant chromosome/particle, hence risks of early convergence is less, but still there might be influence of the chromosomes with best fitness that iteratively involve in reproduction.

Sub-grouping strategy would be the counter option to delay the early convergence caused by dominating particles [10]. Sub-grouping strategy could be performed on both homogeneous and heterogeneous population. In homogeneous sub-group each solution uses same parameters and same operators. In heterogeneous sub-group each solution have different operator and parameters to introduce diversity. Another counter measure to reduce the early convergence is to choose one particle in the swarm as gBest during every iteration and regenerate all or part of other particles. This enables the chance on reducing some diversity to the gBest.

D. Exploring the solution space

GA operators cannot produce all potential solutions as the density of the solution space generated by the population is less. Since all the possibilities for a perfect solution are not explored, the algorithm might miss a better solution than the proposed optimal. Likewise in PSO, the particles cannot move through all possible solutions but the exploitation of the solution space is much higher than GA due to the velocity factor. In both the cases, the drawback could be nullified by introducing local search. The algorithm shall search for a better solution within a certain range of the current proposed solution (particle/chromosome). This process of local search increases the time complexity, hence used with consideration on resources and cost factor.

V. Hybrid approaches

To showcase the power of hybrid approaches, we've compiled a few practical applications of studies that propose PSO and GA hybrid algorithms. This way it's possible to see the range of uses that this method could improve.

A. Integrating particle swarm optimization with genetic algorithms for solving nonlinear optimization problems [11]

The first example aims to apply hybridization for solving non linear optimization problems, and for this, it uses only the objective function and can deal with non-smooth, non-continuous and non-differentiable functions which are actually existing in practical optimization problems. In the proposed hybrid approach particles are initialised, then a loop of PSO and GA is used to improve the particles. Where particles are first improved by PSO and then fed to GA, which evolves them and feeds the particles back to PSO. This approach was found successful in finding the global optima of 17 classical test functions.

B. A hybrid genetic algorithm and particle swarm optimization for multimodal functions [12]

Similar to the previous approach, the authors of [12] used a loop of GA and PSO to use the best aspects of both models. Their findings claim that GA-PSO is a simple but powerful approach that can handle different kinds of continuous optimization problems.

The simulated experiments for the optimization of nonlinear multimodal functions show that GA-PSO is superior to the other methods (continuous hybrid algorithm (CHA), Enhanced continuous tabu search (ECTS), Continuous genetic algorithm (CGA),Enhanced simulated annealing (ESA), continuous reactive tabu search (CRTS minimum)) in the ability to finding the global optimum.

C. Feature Selection [13]

To tackle the issue of feature selection in high dimensional data, the authors in [13] proposed the use of a hybrid of PSO and GA for feature selection for a Support Vector Machine classifier on the Indian pines hyper spectral dataset.

The idea is to overcome both PSO and GA shortcomings, that is, for PSO: the premature convergence of a swarm, and for GA: if a chromosome is not selected, this information will be lost, in addition to that it has a difficulty for finding an exact solution.

The hybrid approach works by integrating the velocity and update rules of PSO with the selection, crossover and mutation steps from GA. The process initiates with n random genes (or particles) which are ranked based on their fitness, in this case, is the Overall Accuracy on SVM validation samples. Then the top half of particles are selected and enhanced by PSO. The other half is populated by performing crossover between individuals in the top half. Finally a mutation occurs to the new individuals with a probability of *0.01*. The method will stop automatically when the difference between the fitness of best solution and the average fitness value of a swarm is less than a predefined threshold.

For comparison, the overall accuracy, the average accuracy, the kappa coefficient and CPU processing time were considered. Among all methods tested (SVM with 220 features, SVM with PSO, SVM with GA, and two feature selection methods), the hybrid approach outperformed all other methods, with a feasible CPU time. From the study it was concluded that the proposed hybrid method can find important features in terms of classification accuracies in an acceptable CPU time. There is also no need to set number of output features using this method, nor initialize any parameters. And due to its evolutionary aspect, it works faster than exhaustive search and therefore can be used appropriately in situation where other feature selection methods can not. The authors also found that this method can also handle high dimensional data with a limited number of training samples, since it uses SVM as a fitness function.

D. Task Scheduling of Cloud Computing [14]

Effective task scheduling of massive task on large cloud platforms is essential to keep up the competitiveness of companies. The authors in [14] analyze the application of a hybrid of PSO and GA to optimize this task. The problem refers to the mapping of user submitted tasks to specific virtual machines, in order to allocate the resources in a way to improve utilization. For each task the goal is to find an appropriate virtual machine to ensure the task execution time is minimized. To facilitate simulation a few assumptions were made: ignore the influence of bandwidth, data transmission, communication time between virtual machines, etc. The task is represented as units of million instructions, and the length is randomly picked from a range. The virtual machines are measured in MIPS (millions of instructions per second), and performance is selected from a range. Finally there are no interdependence between tasks and when multiple tasks are assigned to the same virtual machine they are executed sequentially.

The hybrid approach is a PSO-GA mix based on phagocytosis. Phagocytosis is a defense mechanism of organisms, where phagocyte cells can ingest phatogen cell/bacteria. The fitness function is defined as 1 divided by the max time to complete all

tasks in all machines. The phagocytosis happens in such way that particles are extracted from phagocytic and pathogenic sub populations based on their fitness, and the coding sequence of those individuals are segmented. Then the segments from the particle with high fitness phagocytize the bad particle sequence fragments with low fitness, producing a new individual.

The simulation used a java based cloud simulation named Cloudsim, and with four different experiments (ranging the number of tasks submitted, Virtual Machine's available and number of iterations), the study found that the proposed algorithms can achieve lower time spans for the task allocating when compared to other scheduling methods.

E. Supply Chain Model [15]

The authors in [15] study the application of bi-level linear programming to supply chain distribution problem by applying a hybrid approach of PSO and GA. Bi-level linear programming is a decentralized planning system to deal with two level of systems, the upper and the lower level, playing the role of leader and follower, respectively. The idea is to apply it to a Supply Chain Management problem, defined as a set of approaches to efficiently integrate supplier, manufacturer and stores so that products are produced and distributed in the right quantities.

The study uses three different hybrid approaches. The first one where only the better particles are selected to implement crossover and mutation. The second, all particles are updated using PSO and and followed by crossover and mutation. The third, only the particles which do not evolve are mutated.

The three methods were validated by using four examples adapted from the literature used in the study. The experiments concluded that all hybrid approaches were superior than using standalone methods. Among the hybrid approaches, the third method was the best. The authors highlight that the method is suitable only for bi-level programming and not multi-level programming.

VI. Simulation

To better understand the effects of hybrid approaches, we designed a small experiment which compares the standalone approaches with hybrid ones. The problem explored is the Rastrigin function [16], described in equation 1, an equation famous for having multiple local minimas, see in figure 3, making it a good benchmark for optimization problems. For the equation, we used *A=10* and *d=5*, and the variable we tried to optimize is a five dimension *X*.

$$f(x) = A * d + \sum x_i^2 - A * \cos(2\pi x_i) \quad (1)$$

Some fixed parameters has to be used to facilitate the comparison of results between methods. For every method, only 50 epochs were used, mutation rate was always set to *0.1*, number of population of GA and PSO varies depending on the approach.

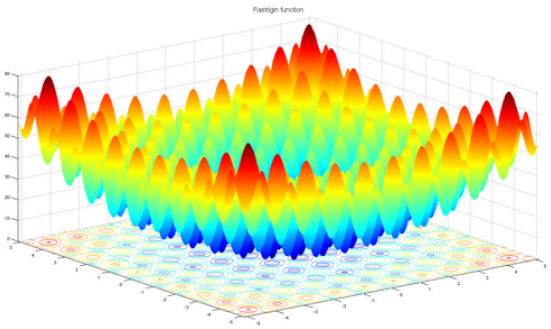

Figure 3: Rastrigin function 3D representation [16].

A. Standalone Genetic Algorithm

The definition of an individual is a 5 dimension array with values randomly initialized in the interval *[-5.12,5.12]* with some noise, the population contains *100* individuals. The fitness function tries to optimize the Rastrigin function, by finding its lowest value. Reproduction is done by tournament selection, and the offspring between two individuals is created by a one point cross-over on a randomly selected point. Mutation is added with a fixed probability by adding small Gaussian noise to the values. Although this algorithm can run for around one thousand epochs in little time, we limited for only 50 epochs for comparison measurements.

B. Standalone Particle Swarm Optimization

The particles were evaluated based on the Rastrigin function and updating their positions based on the function described in equation 2, where W is the inertial weight, and c1 and c2 are acceleration coefficients that influences speed - the default values for these parameters were *0.5*, *1* and *2*, respectively - and the variables r1 and r2 are Gaussian noise. Their position refers to the 5 dimension array of X, which is the variable we try to optimize. In this approach the population size was only *30* individuals. In the equation 2, v stands for velocities, bp stands for particle's best, pp for particle position, and sbp for swarm best particle.

$$v = w * v + c1 * r1 * (bp - pp) + c2 * r2 * (sbp - pp) \quad (2)$$

C. Genetic Algorithm for parameters of Particle Swarm Optimization

This approach used GA to find the parameters that would support finding the best results for the PSO algorithm. In this case, an individual is a 3 dimension array containing values for PSO's parameters, W, C1 and C2, with a population size equal to *30*. The fitness function is the best result found by the PSO algorithm. For every epoch, PSO runs for each individuals, therefore the computational time for this approach is increased, since for a 50 epochs and population size of 30, PSO runs its own 50 epochs a total of 1500 times, resulting in a total of 75000 epochs as per equation 3.

$$total\_epochs = gaEpoch * gaInd * psoEpoch \quad (3)$$

D. Genetic Algorithm for individuals initialization of Particle Swarm Optimization

This approach uses the population of the last generation of the GA algorithm as the initial particles for PSO. So instead of having PSO with randomly generated individuals, it uses instead a set of higher quality individuals filtered by the Genetic Algorithm to achieve better results. The GA runs for 50 epochs, and then the last population is used as initial population of PSO algorithm, which also runs for 50 epochs, for this approach *100* individuals were used per generation. The fitness function measures only the results from PSO.

E. Alternating Genetic Algorithm and Particle Swarm Optimization

Taking a step further from the previous approach, this loops between GA and PSO, by starting with GA, and feeds the following algorithm its best set of individuals. The results measure the best individual/fitness from the end of each loop. In this method, one epoch counts as the run of GA and PSO together - each running 50 epochs - with a total of 100 epochs per loop and 5000 epochs in total. A population size of *100* individuals was used.

F. Results

Analysing the picture 4, we can see the evolution of best fitness over each epoch for all the five methods. One important thing to notice is the initialization of these methods, as the standalone methods have worse initialization when compared to hybrid methods.

PSO and PSO with GA individuals behave similarly, as it's expected, but the hybrid approach starts with much better results due to the GA individual initialization. In the end, both approaches achieve quite similar results.

The two methods that yield the best results in this limited epoch frame are the *PSO with GA parameters* and *PSO with GA alternating*, achieving values close to *1.0*, a significant gap from all the other methods.

Furthermore we can take a look at the computational time required for each method to run the 50 epochs. In picture 5 it's clear the drastic gap between the *PSO with GA parameters* and all the others, due to its large amount of repetitions, followed by *PSO with GA alternating*. Since the two best methods were the ones that had more repetitions due to their nature, it could have had an influence in their advantage.

By limiting the amount of epochs we're capable to make an easier comparison of the method's evolution, but it's not necessarily the best, as some algorithms may take longer time to converge to a better solution. Therefore we decided also to perform a comparison based on feasible time of 1 minute, and let every algorithm run until then and finally compare results. In figure 6 we can see that even using the same computational time, the methods *PSO with GA parameters* and *PSO with GA alternating* still outperformed the others.

To summarize our achievements, both comparison approaches were repeated 10 times, and the averaged best results were recorded for a more robust analysis. In table I we can see that GA is the worst performing algorithm, PSO and PSO with GA individual initialization perform similarly, given the time, since the only advantage of the hybrid approach in this

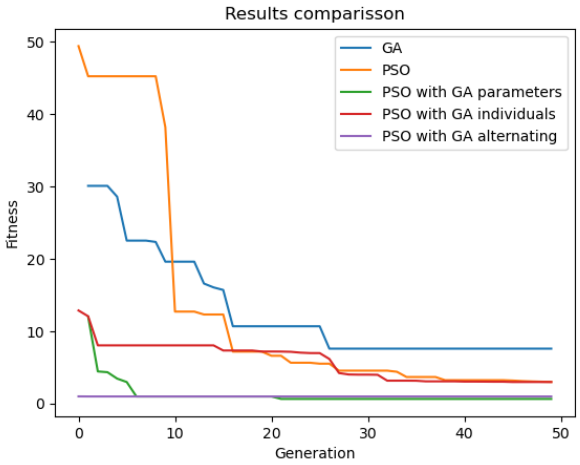

Figure 4: Comparison of results of the five methods using a fixed amount of epoch equal to 50.

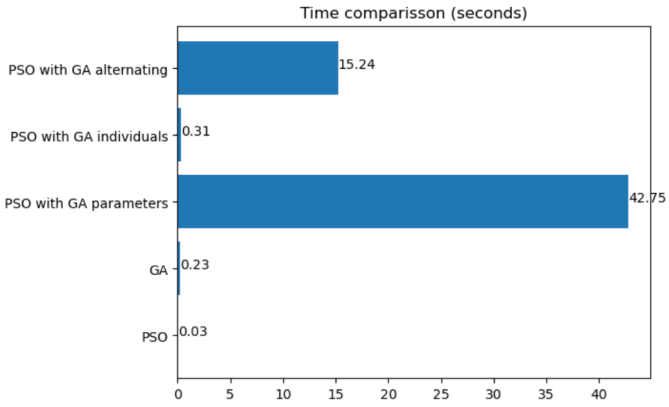

Figure 5: Comparison of computational time of the five methods using a fixed amount of epoch equal to 50.

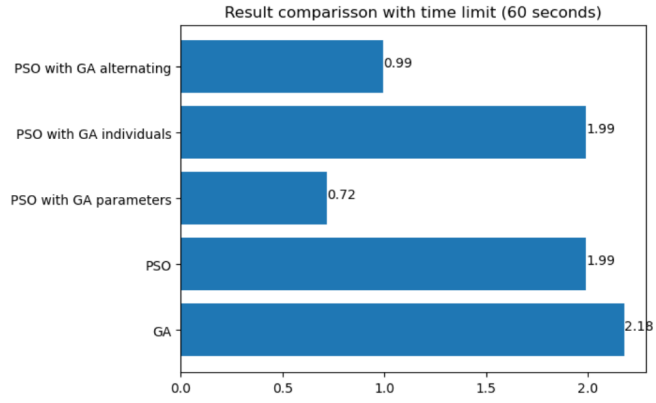

Figure 6: Comparison of results of the five methods using a time constraint of 1 minute.

case is a better starting point. In every case, PSO with GA parameters yields the best results, closely followed by PSO with GA alternating.

This simulation works to further demonstrate the power of hybrid approaches over standalone ones, even for straightforward problem such as the global minima for Rastrigin function.

Methods	50 ep	50 ep (avg)	1 min	1 min (avg)
GA	7.604	10.225	2.178	3.590
PSO	3.007	4.298	1.990	4.676
PSO GA parameters	0.655	1.819	0.719	0.938
PSO GA individuals	2.993	2.750	1.990	1.592
PSO GA alternating	0.995	2.288	0.995	1.094

Table I: Summary of results of simulation using the Rastrigin Function.

VII. Applying a hybrid approach for image clustering

In section VI we saw in practice the effectiveness of hybrid swarm approaches, where combining Genetic Algorithm to find the best hyperparameters for swarm algorithms - in our case, Particle Swarm Optimization - was found the most effective. This section explores Ant Clustering, another kind of Swarm Intelligence based on chemical recognition of ants, and goes over an application of this algorithm for clustering images of cars [1]. Furthermore, the hybrid approach with GA is applied to the AntClust application with the goal of finding best results.

A. AntClust

The authors in [1] apply the method proposed in [2] to cluster images. The AntClust is a clustering algorithm based on chemical recognition system of ants to recognize their nest mates and enemies based on their odor. In this representation, each ant represents a data tuple (in this case, an image). A few concepts that happen in nature are represented in this paper. Each ant has an individual odor, and each ant maintains an odor template based on their encounters with other ants, if their odors are similar enough they will know that both ants are from the same colony, rejection can happen if odors are too different. Each ant has its own odor template and is constantly updated by encounters with other ant, since there is no global odor template this algorithm functions as decentralized.

For the application in [1], the dataset used was the vehicle re-identification dataset. And the cluster formed by the ants colony are formed by different images of a distinct car. An ant has the following characteristic: its genetics (data tuple), a label referring to which colony it belongs to, an age to refer how many meetings it had, an estimator of success of meetings, an estimator of how accepted an ant is in its own colony, the mean similarity between this ant and all the ones it has met, and a maximal similarity. The algorithm has three distinct phases.

In phase one all parameters are unitized accordingly: the gene will be the i-th tuple of data, and all other parameters will be zero. Phase two initializes the template, by having every ant meet randomly selected ants. Phase three picks two random ants and they meet. During a meeting, the following rules are applied:

- R1: - if same label (no colony) and accepted, create new label and assign it to ants. If not accepted, apply R6
- R2: If different labels and accepted, but one ant already has a colony, while the other does not, then second ant is assigned to that colony.
- R3: If same label (with colony) and accepted, increase success estimator in both ants.
- R4: if same label (colony) and not accepted, decrease success estimator of both ants. Additionally, the ant with smaller within colony success estimator loses its label (doesn't belong to colony anymore).
- R5: If different labels and accepted, decrease success estimator and the ant with lower estimator is assigned to the other's colony.
- R6: if no other rule is applied, nothing happens.

At first small clusters will form and later grown in size as the ants meetings proceed. The Acceptance function will return true or false based on the similarity of the ants in relation to each ant's template. The implementation keeps flexibility by defining two classes: AntClust and Ant, and two interfaces: similarity_interface and rule_interface. AntClust class deals with the algorithm and meetings of ants. Ants class deals with parameters of each ant. Similarity interface defined the similarity function. Rule interface define the rule set.

For our adaptation we focus on the rule-set, where rules are defined in a file called rules.py, as well as the AntClust class. The authors found that AntClust achieves good results for small number of clusters, without the need to previously know the number of cluster, such as the K-mean approach. The authors discuss that the performance is subjective to the rule set, and that, although complex, could be improved via others bio inspired approaches, such as Genetic Algorithm. The authors of [1] tested the algorithm first for random generated numbers, then for the image data, we only perform the testing for the numbers data, due to limitations when dealing with the image dataset.

B. GA and AntClust

This section will explain our hybrid adaptation of the previously described algorithm and discuss the results and implications found.

1) GA for AntClust Hyper-parameters: We observed that the AntClust algorithm also depended on a set of parameters that defined the performance, which was much simpler to apply the GA to. Those are:

- Alpha: based on this the meetings for the first phase will be calculated by 0.5*alpha*Num_data_points
- Betta: used to calculate how many other ants a single ant will meet to initialize their template.

- Shrink: this value will be used for the probabilistic nest fitness calculation in nest_shrink.
- Removal: the value is a floating point percent value giving the fitness threshold for deleting a nest with a smaller fitness

For testing the authors used the Adjusted Rand Index (ARI) score is a function that measures similarity, commonly used in clustering algorithms [17][18]. Let a be the numbers of pairs of elements that are in same set in the cluster prediction vs the real labels. And b the numbers of pairs of elements that are in different set in the cluster prediction vs the real labels. And C the total number of possible pairs in the dataset. The ARI is $(a + b)/c$. The ARI is a corrected-for-chance version of RI.

The GA uses only 10 individual and 10 generations, and the fitness function is measured by the mean Adjusted Rand Index score of the clustering following the test methodology applied in [1], with following parameters:

- cluster_width = 0.1
- clusters_min = 2
- clusters_max = 4
- values_per_cluster_min = 3
- values_per_cluster_max = 90
- tests_per_clusters = 5

Due to time and computational constraints it was necessary to limit the amount of tests per cluster, and since the results found in [1] found that from 5 clusters onward the efficiency goes down, we limited ourselves to test between 2 to 4 clusters. The values per cluster and cluster width remained unchanged.

To compare the results we measured the best parameter set found by our Genetic Algorithm against the parameter used by the authors of [1] during testing:

- **Standard parameters**:
  - *alpha*: 500
  - *betta*: 0.9
  - *shrink*: 0.2
  - *removal*: 0.3
- **GA parameters**:
  - *alpha*: 372
  - *betta*: 0.95060
  - *shrink*: 0.46611
  - *removal*: 0.32037

After achieving the best parameter set out of the GA, we tested for number of cluster from 2 to 10, and compared the results - which can be seen in table II on appendix. Out of the 9 tests, GA outperformed the fixed parameter in 5 runs - for 2, 5, 6, 7 and 10 clusters. Whereas the Fixed parameter outperformed on 3 runs - 3, 8 and 9 clusters. Both approaches scored the highest for the number of clusters 4.

Figures 8 and 7 show the behavior of the GA parameter the fixed, respectively. It's important to highlight, due to limitations, the GA ran only for number of clusters 2 to 4, therefore the parameter used may not be optimal for subsequent number of clusters, yet still outperforms the fixed parameters for four out of six of these higher number of clusters.

2) GA for AntClust Rule sets: The definition of a rule set is a combination of rules (without specific order) that leads to the clustering of similar individuals. These rules consist of two parts: condition and consequence. A rule set has a much higher impact when compared to the parameter discussed in section VII-B1. If the rule set doesn't make a logical sense in the context of the AntClust algorithm, there may not be any clusters at all. So it is important that when finding a representation of a gene for the rule set, that it's done in a way the population can evolve and have better results in time.

We propose and test a possible schema that could help guiding this implementation in future works. The idea is to break the original rule set from Labroche [2] into smaller blocks that can be combined in different permutation that could lead to functional and better rule sets. First we define objects of the class conditions:

- Acceptance (ants accept each other based on similarity).
- Rejection (ants reject each other based on similarity).
- Ants have same label.
- Ants have different labels.
- Both Ants have labels (belong to a colony).
- Both Ants don't have labels.
- One ant has no label, the other does.
- AND Boolean operator

Then we define objects of the class consequences:

- Create new label and assign to both ants.
- Assign existing label of ant to unlabeled ant.
- Increase meeting estimators of both ants.
- Decrease meeting estimators of both ants.
- Remove ant with lower meeting estimator from colony (label) and reset parameters.
- Add ant with lower meeting estimators to the other ant's colony (label).
- Do nothing

With these set of conditions and consequences it would be possible to randomly generate a rule set by creating a sequence of condition-consequence pairs. One example would be:

- Condition : ["Acceptance","and", "not", "same labels"].
- Consequence: ["Decrease meeting estimators", "Remove ant with lower meeting estimator from colony"]

The generation of new individuals was produced via crossover by randomly selecting one point in the list of rules and crossing each part from its respective parent. Mutation was used to change a randomly selected rule by generating a completely new condition and consequence. Another possible implementation would be switching one piece of condition/consequence list by another condition/consequence object.

Fitness was measured the same as described in section VII-B1: by running the AntClust for a few number of clusters and returning the mean ARI. We executed the GA algorithm for the rule set, with 30 individuals and 20 generations, but with a random initialization of individuals described previously, the algorithm did not manage to provide any functional rule set, leading all mean ARI equal to zero.

Although this couldn't provide any valid rule set, we believe it could work as the base building block for future work, where the initialization of rule set could follow a more complex structure to develop functional rules. One example could be the sampling from the consequence list without replacement, so that each consequence would be used only once. Another improvement could be limiting the use of contradictory conditions for the same rule (not allowing conditions such as "acceptance and not acceptance").

By having an completely random initialization, we estimate that the amount of generations and population size necessary to achieve only a few functional rule set would be much higher than the necessary to find better hyperparameter. This would exceed our computational and temporal resources.

VIII. Conclusion

Based on the research made, it's possible to infer the superiority when applying hybridization to swarm intelligence algorithms. In our study we showcase this success when applied specifically to Particle Swarm Optimization and Genetic Algorithms, in our practical simulation the advantage of hybrid approach was considerate even for a simple problem such as the Rastrigin function.

Furthermore other algorithms can benefit from this principle, such as the AntClust algorithm. For this we applied Genetic Algorithm to the hyper-parameters of the AntClust class and even with little computational resources and time, we could see a slight improvement in the results. We speculate based on research and application that given more resources to run the GA for longer and using longer tests for the fitness function, it could have significant impact in the AntClust method, making it a robust clustering algorithm without the need for manual hyper parameter tuning.

For further work, as mentioned in [1], applying GA to a rule set could improve significantly the performance of AntClust. Unfortunately, due to the nature of the rule set, representing it into an individual's gene - capable of random initialization, feasible reproduction and mutation, and evolution in a way that produces actually *better* rule sets - would require extended research and experimentation. We offer another idea, of finding better hyperparameters of the AntClust class and we hope that our work for the rule set GA can serve as a base for future research to develop better implementation for [1]. The codes for the simulation can be found in https://github.com/ericsiqueira99/SelfOrganizingNetworks. The code for the AntClust implementation can be found in https://github.com/ericsiqueira99/AntClustGA

References

[1] Gero Oed and Memarmoshrefi. Testing the performance of antclust: a clustering algorithm based on the chemical recognition system of ants. 2022.

[2] Nicolas Labroche, Nicolas E, and Gilles Venturini. A new clustering algorithm based on the chemical recognition system of ants. 02 2003.

[3] Peggy WRIGHT. Knowledge discovery in databases: tools and techniques. *The ACM Magazine for Students*, 5(2):23–26, 1998.

[4] Yun Shin Koh Patricia Riddle Saeed Ur Rahman shafiq Alam, Gilian Dobbie. Research on particle swarm optimization based clustering: A systematic review of literature and techniques. *Swarm and Evolutionary Computation*, 17(1):1–13, 2014.

[5] Amrita Chakraborty and Arpan Kar. *Swarm Intelligence: A Review of Algorithms*, pages 475–494. 03 2017.

[6] Manoj Kumar, Mohamed Husain, Naveen Upreti, and Deepti Gupta. Genetic algorithm: Review and application. *SSRN Electronic Journal*, 2010.

[7] DR Ramdania, M Irfan, F Alfarisi, and D Nuraiman. Comparison of genetic algorithms and particle swarm optimization (pso) algorithms in course scheduling. In *Journal of Physics: Conference Series*, volume 1402, page 022079. IOP Publishing, 2019.

[8] Tom V Mathew. Genetic algorithm. *Report submitted at IIT Bombay*, page 53, 2012.

[9] Junyan Chen and Huiying Zhang. Research on application of clustering algorithm based on pso for the web usage pattern. In *2007 International Conference on Wireless Communications, Networking and Mobile Computing*, pages 3705–3708. IEEE, 2007.

[10] Voratas Kachitvichyanukul. Comparison of three evolutionary algorithms: Ga, pso, and de. *Industrial Engineering and Management Systems*, 11(3):215–223, 2012.

[11] W.F. Abd-El-Wahed, A.A. Mousa, and M.A. El-Shorbagy. Integrating particle swarm optimization with genetic algorithms for solving nonlinear optimization problems. *Journal of Computational and Applied Mathematics*, 235(5):1446–1453, 2011.

[12] Yi-Tung Kao and Erwie Zahara. A hybrid genetic algorithm and particle swarm optimization for multimodal functions. *Applied Soft Computing*, 8(2):849–857, 2008.

[13] Pedram Ghamisi and Jon Atli Benediktsson. Feature selection based on hybridization of genetic algorithm and particle swarm optimization. *IEEE Geoscience and Remote Sensing Letters*, 12(2):309–313, 2015.

[14] Xueliang Fu, Yang Sun, Haifang Wang, and Honghui Li. Task scheduling of cloud computing based on hybrid particle swarm algorithm and genetic algorithm. *Cluster Computing*, 26(5):2479–2488, May 2021.

[15] R.J. Kuo and Y.S. Han. A hybrid of genetic algorithm and particle swarm optimization for solving bi-level linear programming problem – a case study on supply chain model. *Applied Mathematical Modelling*, 35(8):3905–3917, 2011.

[16] Eric Benhamou, David Saltiel, Beatrice Guez, and Nicolas Paris. Bcma-es ii: revisiting bayesian cma-es. 04 2019.

[17] Douglas Steinley. Properties of the hubert-arable adjusted rand index. *Psychological Methods*, 9(3):386–396, 2004.

[18] Lawrence Hubert and Phipps Arabie. Comparing partitions. *Journal of Classification*, 2(1):193–218, December 1985.

APPENDIX

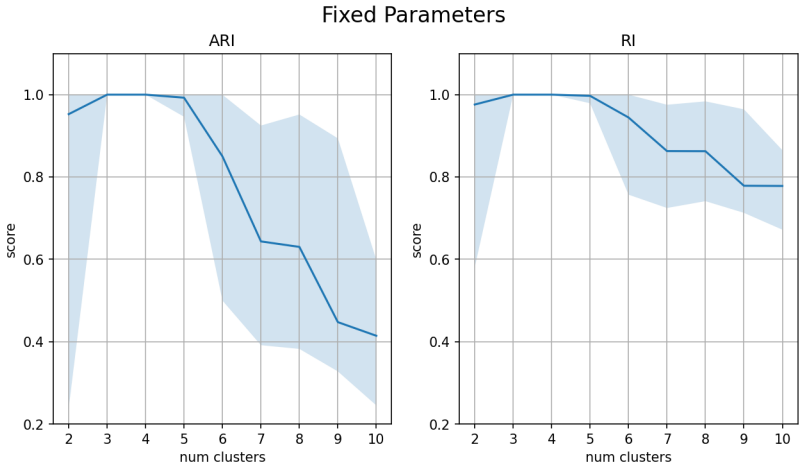

Figure 7: Behavior of AntClust for number of cluster from 2 to 10, using the fixed parameter used by authors in [1].

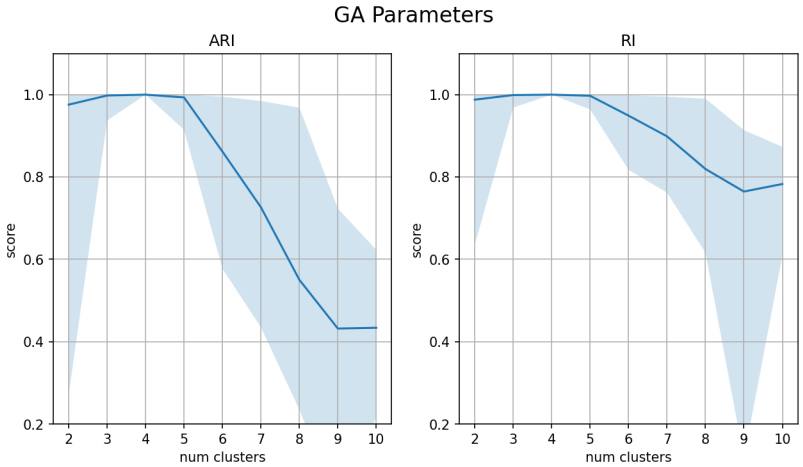

Figure 8: Behavior of AntClust for number of cluster from 2 to 10, using the parameter optimized by GA.

Num clusters	Fixed ARI	Fixed RI	GA ARI	GA RI	Best ARI	Best RI
2	0.952589	0.975984	0.975695	0.987834	GA	GA
3	1.000000	1.000000	0.997913	0.998956	Fixed	Fixed
4	1.000000	1.000000	1.000000	1.000000	Equal	Equal
5	0.992561	0.997084	0.993272	0.997286	GA	GA
6	0.849935	0.944469	0.861555	0.948985	GA	GA
7	0.643491	0.862913	0.726394	0.898753	GA	GA
8	0.630180	0.862628	0.550109	0.819406	Fixed	Fixed
9	0.447469	0.778694	0.431815	0.764610	Fixed	Fixed
10	0.414322	0.778201	0.433749	0.782941	GA	GA

Table II: Results of AntClust application to number data using default parameters used for test by authors of [1] - namely "Fixed" - and parameters found by our Genetic Algorithm approach - namely "GA".