[[1]](#footnote-1)

Adaptive Algorithm for Improving Particle Swarm Clustering Convergence Rates

Jared H McLean, CS 422, April 27, 2017

*Abstract*— This paper will compare various clustering algorithms, focusing on k-means and particle swarm clustering techniques and propose a potential approach for increasing the convergence rate and stability of particle swarm based clustering algorithms. A total of 5 clustering algorithms will be developed and compared, k-means, particle swarm optimization clustering, k-means seeded particle swarm clustering, an adaptive particle swarm based approach that attempts to increase convergence rates by evolving the algorithm’s acceleration coefficients, and a k-means seeded variation of this adaptive algorithm. The algorithms will be tested against 4 data sets, the iris dataset, BUPA dataset, Car dataset, and wine dataset. Results will be quantified by four factors, overall fitness of the final clustering, number of iterations take for convergence, inter-cluster distance, and intra-cluster distance. In general, while improving the convergence rate of the particle swarm clustering algorithm, the adaptive approach performs very poorly when not supplemented by the k-means seeded centroid population. The k-means seeded variation, while performing well and increasing convergence rates over the seeded normal particle swarm approach, the increased overhead and may not warrant the gains. A similar heuristic, however, may allow greater stability for particle swarm clustering and foster convergence in a less aggressive manner.

# INTRODUCTION

C

lustering is one of the most major concepts in data mining applications. Under many circumstances, creating an algorithm that can group together unknown data without having to be trained against the specific set is a necessary task. In addition to simple effectiveness of such an algorithm, computational efficiency of clustering algorithms can be an issue, particularly for very large or time sensitive data sets. Unfortunately, in many cases, there tends to be a tradeoff between computational efficiency and clustering effectiveness. The k-means algorithm, for example, is very simple and computationally efficient; however, the clusters it produces are often not as good as other algorithms. Due to its simplicity and efficiency, though, k-means can be used as an effective groundwork for the development of other techniques. This paper will review some of the techniques designed to build upon the basic concepts of k-means clusterings, specifically modifications based on the concepts of particle swarm optimization, or PSO, and propose a potential heuristic for improving the convergence rate of these techniques.

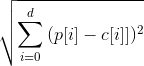
# Literature Review and Background

## K-Means Clustering

K-Means clustering serves the foundation for the clustering techniques used in this study. K-Means clusters data points in a data set to a group of centroids. Each step of the procedure follows the procedure:

1. For data point, p, in the data set
   1. Assign p to the cluster of the nearest centroid using equation 1
2. For each centroid, c
   1. Move c to the center of its cluster via equation 2

This procedure is repeated until some convergence criteria is met. Due to the final position of k-means clustering being highly dependent on its initial clusters, the algorithm is highly dependent on its starting positions. While heuristics can be applied to attempt to find a “good” starting point for the algorithm the basic form of this clustering technique will use random centroids to begin, which may lead to subpar results in cases where the initial centroids are located in poor positions [1]. In spite of the drawbacks to k-means, it has proven to be an influential algorithm in terms of clustering techniques, largely due to its simplicity and rapid convergence rates [4]. Resultantly, it has also been the subject of many spin-off algorithms attempting to improve upon its accuracy while maintaining its quick convergence rates.



Equation 1: Where p is the data point being clustered, c is a centroid to be compared to, and d is the dimensionality of the data set

C:\Users\Jard\Downloads\CodeCogsEqn (7).png

Equation 2: Where p is the set of data points in the current

cluster, c is a centroid to be compared to, d is the dimensionality of the data set, and m is the number of points in the current cluster

## Particle Swarm Optimization Based Clustering

Particle swarm optimization based clustering, henceforth referred to as PSO clustering, is a technique for improving upon the accuracy of k-means clustering using several populations of centroids as particles, and updating the particles based on how well the clustering does based on some fitness function [1]. Particle swarm optimization is a stochastic method for optimizing a problem set by using principles laid out by nature – the socio-behavioral patterns of birds and other swarm animals [5]. The general procedure for performing particle swarm optimization is as follows:

1. For each particle, p
   1. Evaluate the particle’s fitness using some fitness function
   2. If the current fitness of the particle exceeds that of the currently stored global optimum, gbest, assign the particles current location to gbest
   3. If the current fitness of the particle exceeds that of the currently stored local optimum, pbest, assign the particles current location to pbest
   4. Update the particle’s velocity and position the using equations
      1. v[] = x \* (v[] + c1 \* rand() \* (pbest[] - present[]) + c2 \* rand() \* (gbest[] - present[]))  
         OR  
         v[] = w \* v[] + c1 \* rand() \* (pbest[] - present[]) + c2 \* rand() \* (gbest[] - present[])
      2. present[] = present[] + v[]

Where v[] is the velocity vector of the particle, present[] is the current particle location, gbest[] and pbest[] are the global population and local particle optima respectively, rand() is a function returning a random number between 0 and 1, c1 and c2 are learning factors known as acceleration coefficients, x is a constriction coefficient defined by equation 3, and w is a user defined inertial weight less than one [6]. This procedure is repeated until some stopping criteria is met, typically a maximum number of iterations or a change in population fitness below some defined threshold over some number of iterations is observed [1]. In the context of PSO clustering, each particle has its own subpopulation of centroids that move independently of one another. Each centroid will have its own velocity vector, and location; however, pbest and gbest are determined as a function of the individual particles as a whole. Hence, each population of centroids will store a set of positions for each centroid as its personal best position, and gbest will hold the set of centroid locations that has the best fitness among these individual centroid populations. As such, centroids will be updated based on the corresponding position vectors in pbest and gbest. That is, the first centroid in each centroid population will correspond to the first centroid in every other population for the sake of determining the sub-vector in gbest to use for evaluating update functions. As in k-means clustering, randomized centroid positions will serve as the starting point for each population of centroids. PSO clustering tends to provide better results than k-means, but takes much longer to converge [1], as is an issue with many evolutionary renditions of the k-means algorithm [4].

C:\Users\Jard\Downloads\CodeCogsEqn (10).png

Equation 3: Where c = c1 + c2 and c <= 4

## K-Means Seeded Variations on PSO Clustering

This technique uses an instance of the k-means clustering algorithm to seed a population of centroids for a PSO based clustering algorithm. The general procedure for this is as follows:

1. Generate n random centroid populations for use by a PSO clustering algorithm
2. For the first centroid population in the generated set apply k-means clustering
3. Run the PSO clustering algorithm on the seeded set of centroid populations

This modification should, theoretically, allow the algorithm to converge faster by providing a partial solution for the algorithm to converge on; thus, drawing the particles towards a plausible search space rather than an entirely random one [1]. In addition to general efficiency improvements, this technique also ensures that the clusterings will be no worse than the k-means algorithm, since no cluster worse than the initial population of centroids seeded by the k-means algorithm can be accepted as the global best population without improvements. By converging from random positions to the centroids found by the k-means algorithm, particles are also able to traverse the surrounding space for an improved solution that may have been missed by the k-means algorithm. This reduces the impact of the algorithms’ sensitivity to starting locations, since finding an optimal starting position is akin to creating an optimal clustering – an NP-hard problem [2].

# Proposed Algorithm

## Adaptive PSO Clustering

One of the primary factors in the convergence of a PSO algorithm is the selection of c values and the corresponding inertial weight or constriction coefficient. Additionally, if using a constriction coefficient, one of the issues with constriction coefficients in some cases is a static value can cause unwanted fluctuations in the population after it should have converged, leading to potentially higher convergence iteration [3]. The proposed algorithm would attempt to counter these issues by removing the static nature of the c values. An adaptive approach, updating c values dynamically, will allow adjustments to the c values in such a way as to foster a quicker convergence on a solution and increase the stability of the final population. The proposed method for achieving this is the use of genetic algorithms to determine c values for each generation of the PSO algorithm. At each generation, the current population of c values will be modified by a genetic algorithm attempting to minimize the velocity of the particles. The fitness of the individuals will be determined by the magnitude of the velocity vector using equation 4.



Equation 4: Where v is the velocity vector for the individual and d is the dimensionality of the vector

# Implementation

The discussed clustering algorithms, as well as a k-means seeded version of the proposed adaptive algorithm, were implemented using python, and the number of iterations required for convergence, average interclustular and intraclustular distances, and fitness function of the final clustering were recorded. For all algorithms, the clustering fitness was determined using the average of the fitness of each individual centroid’s cluster. The fitness of individual clusters was determined by taking the average of the sum of square distances for each data point in the cluster – where distance was determined by the Euclidian distance equation defined by equation 1. Clusters that contained no data points were excluded from this calculation, and their centroids were not considered when taking the average of the cluster fitnesses. For each algorithm, a k value – the number of centroids in each centroid population – of 4 was used. The pseudo-random number generator used for producing random values for the procedures was the operating system’s built in random number generator using the random function from the python os library. While OS dependent, this source is expected to be sufficient for cryptographic use [7]. Since PSO and GAs rely heavily on random number generation, it is important to have a sufficient random number generator. Furthermore, all data values were converted to the range -1 to 1 to ensure proper classification of attributes with different domains and ranges [1].

## K-Means Clustering Details

The k-means algorithm was implemented as defined by the earlier discussion. Convergence was defined when there was no change in the fitness of the clustering, when the fitness of the clustering was equal to that of the clustering two iterations ago, or when 50 iterations was reached. The second condition ensures that if the algorithm gets stuck in a cycle it will recognize this and terminate.

## PSO Clustering Details

The PSO clustering algorithm was implemented primarily using the values defined by DW van der Merwe and AP Engelbrecht’s paper “Data Clustering using Particle Swarm Optimization” [1]. The learning factors were set as c1 = c2 = 1.49 and an intertial weight w = 0.745 was used. The referenced paper defines w = 0.72; however, this value was modified to fit the equation w = c / 4, where c = c1 + c2. This modification was made in order to provide a function that was continuous at c = 4 with the equation for the constriction coefficient, such that w is used where c < 4 and x where c >= 4. Additionally, the value yielded by this equation for the given c value is very near the inertial weight defined by the paper. The values c1 = c2 = 2 were also attempted to be used, though these values occasionally resulted in a runaway velocity, potentially due to the domain the data was shifted to. A set of 10 particles, or individual centroid populations, was used, and convergence was defined as a less than 2 percent change in the average clustering fitness of these 10 populations over 5 iterations. If this convergence condition was not met, the algorithm would stop after 300 generations.

## Adaptive PSO Clustering Details

C values used by the adaptive algorithm were initially selected at random for each centroid as a value between 0 and 5. Populations were defined as the populations within clustering groups. That is, the genetic algorithm ran for each population of centroids, using the centroids as the chromosomes for evolution. An attempt was made to partition the population across the corresponding centroids between the centroid populations, creating populations in line with those used by the clustering portion of the algorithm; however, it was found that this partitioning yielded slower convergence rates and no improvement in results. Selection for the genetic algorithm used tournament selection with replacement between 3 randomly selected individuals. Tournament selection was chosen as the selection operator in order to reduce the complications that could arise from roulette wheel based selection algorithms where the magnitudes of the velocity vectors were very close, which would likely arise near convergence. Crossover was performed by replacing the parents’ c values by a random number between the c values of the parents. For example, if the first parent’s value c1 = 1.2 and the second parent’s value c1 = 1.3, the crossover operation would replace each of these values by a random number on the range [1.2, 1.3]. Upon selecting parents, a random integer between 0 and 3 inclusive was generated. If the generated value was 0, crossover would be performed on c1, if the value was 1 crossover would be performed on c2, if the value was 2, crossover would be performed on both, and if the value was 3, no crossover would be performed. Mutation was performed with a 1% chance for each c value, and would result in the c value being replaced by a new random number between 0 and 5.

## Seeded Clustering Algorithm Details

The seeded algorithms were implemented as discussed and used all the same parameters as their non-seeded counterparts.

# Experimental Results

the data sets used were the iris dataset, the BUPA dataset, the wine dataset, and the car dataset. These datasets have the following features:

Iris: 4 non-classifier attributes, 3 classifications, 150 records, continuous attributes

BUPA: 6 non-classifier attributes, 2 classifications, 341 records, continuous attributes

Car: 6 non-classifier attributes, 4 classifications, 1728 records, continuous and discreet attributes

Wine: 13 non-classifier attributes, 3 classifications, 178 records, continuous attributes

[9]

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table 1: Average results of clustering algorithms on the four data sets over 30 iterations | | | | |
|  | Iris | BUPA | Car | Wine |
| K Means |  |  |  |  |
| Average Intradistance | 0.36686911919857595 +/- 0.02992332914628612 | 0.4773215493450503 +/- 0.01706820554094675 | 0.5896241335181274 +/- 0.008560107946942636 | 0.9696285141955782 +/- 0.013450401554651121 |
| Average Interdistance | 1.5042910685108886 +/- 0.11354041676223023 | 1.0632995333463875 +/- 0.19687710207576212 | 1.6731451513584878 +/- 0.026637705857444356 | 1.6610629507743073 +/- 0.05744911278866882 |
| Average Convergence Iteration | 9.933333333333334 +/- 2.9992591677871974 | 15.6 +/- 6.750802421440974 | 20.233333333333334 +/- 5.245209454561585 | 11.233333333333333 +/- 3.639444402042097 |
| Average Best Fitness | 0.16038352849254867 +/- 0.028698249805282787 | 0.4055474314519239 +/- 0.03695450337552676 | 2.616548134632532 +/- 0.029858272334579217 | 1.0626744699900954 +/- 0.03214402894241444 |
|  |  |  |  |  |
| PSO |  |  |  |  |
| Average Intradistance | 0.4064830863149433 +/- 0.04263898482414914 | 0.5836688089809727 +/- 0.04031351056590361 | 1.6489748909214172 +/- 0.029643375267715055 | 1.3290779934996337 +/- 0.09704612508228473 |
| Average Interdistance | 1.4940518030328982 +/- 0.1671370466681281 | 0.7680537064316598 +/- 0.5178184729275063 | 1.7112843225196708 +/- 0.16808107242360265 | 1.4494029092119358 +/- 0.6677807392003811 |
| Average Convergence Iteration | 116.5 +/- 43.51225880906054 | 135.63333333333333 +/- 73.35006627278685 | 45.46666666666667 +/- 25.246429890624583 | 124.46666666666667 +/- 49.40832138640436 |
| Average Best Fitness | 0.18249843946975478 +/- 0.037521023799538755 | 0.38642401238652524 +/- 0.0768614876827758 | 2.7144966012963776 +/- 0.08678899908416403 | 1.6476677945389395 +/- 0.27759026214829235 |
|  |  |  |  |  |
| Adaptive PSO: |  |  |  |  |
| Average Intradistance | 0.5741560181988249 +/- 0.08454540540357448 | 0.7553025972459746 +/- 0.11191483629917459 | 1.6541554257437856 +/- 0.024835896566086358 | 1.5527265922995501 +/- 0.10169886838564851 |
| Average Interdistance | 1.3024781655641313 +/- 0.25186717705180817 | 1.0984665188178062 +/- 0.4554519897222369 | 1.548148699385192 +/- 0.1447378808614093 | 1.701639303640627 +/- 0.4056375802638811 |
| Average Convergence Iteration | 78.1 +/- 73.87166800156426 | 78.16666666666667 +/- 79.9850333222132 | 23.833333333333332 +/- 25.419262162558706 | 62.3 +/- 42.763029203585035 |
| Average Best Fitness | 0.34642573199105187 +/- 0.08675018282517878 | 0.6833011946069811 +/- 0.12881502139877857 | 2.7693236395423515 +/- 0.049643934085839676 | 2.339504987522921 +/- 0.2970096366217503 |
|  |  |  |  |  |
| Seeded PSO |  |  |  |  |
| Average Intradistance | 0.36715496649420015 +/- 0.029487857112286575 | 0.47889722627432296 +/- 0.0174454909329435 | 1.590127834313939 +/- 0.00861848285391794 | 0.970456694091919 +/- 0.013824010082619468 |
| Average Interdistance | 1.486081426297133 +/- 0.09641606071352303 | 1.0418901863172598 +/- 0.23745507571715876 | 1.6769304333113 +/- 0.03434869054231797 | 1.6681872449645097 +/- 0.06997102719642145 |
| Average Convergence Iteration | 79.1 +/- 45.01433105134408 | 84.6 +/- 46.53786988392715 | 27.2 +/- 9.689857240090449 | 60.6 +/- 17.88593488377576 |
| Average Best Fitness | 0.1574120776287936 +/- 0.019641920091662535 | 0.3830224903628103 +/- 0.03414152952818456 | 2.6156691904469103 +/- 0.030131694739331454 | 1.0573722213686831 +/- 0.026255938773130958 |
|  |  |  |  |  |
| Seeded Adaptive PSO |  |  |  |  |
| Average Intradistance | 0.3668847410014177 +/- 0.029999564571498022 | 0.47825589304217436 +/- 0.017543016546358967 | 1.5895979793678991 +/- 0.008538753228991908 | 0.9700845930148934 +/- 0.013288500889394287 |
| Average Interdistance | 1.5017936499844957 +/- 0.11545001998217923 | 1.0603673825995314 +/- 0.19910333845643516 | 1.67160346697044 +/- 0.02978750560157981 | 1.664335567975143 +/- 0.05760456669812304 |
| Average Convergence Iteration | 55.3 +/- 57.51414318814692 | 47.43333333333333 +/- 39.46701857951213 | 19.366666666666667 +/- 20.626008392857354 | 57.46666666666667 +/- 59.95650275176349 |
| Average Best Fitness | 0.1602263384486463 +/- 0.02871122976031806 | 0.39883701196722676 +/- 0.03478977872293728 | 2.6145477567684994 +/- 0.02821705821532357 | 1.0604970183226796 +/- 0.03222618212910561 |

Each clustering algorithm was run against each dataset 30 times. The average of each tracked statistic was recorded in table 1.

Additionally, clusterings produced and a graph of the number of iterations vs the fitness of the clustering for the iris data set is shown by figures 1-5

Figure 1: K-means clustering

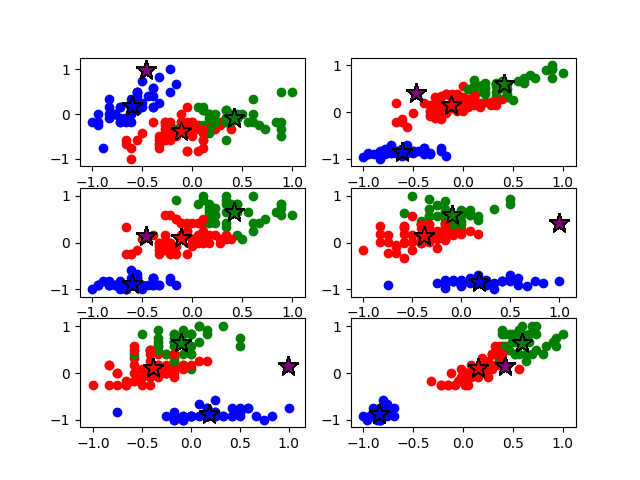
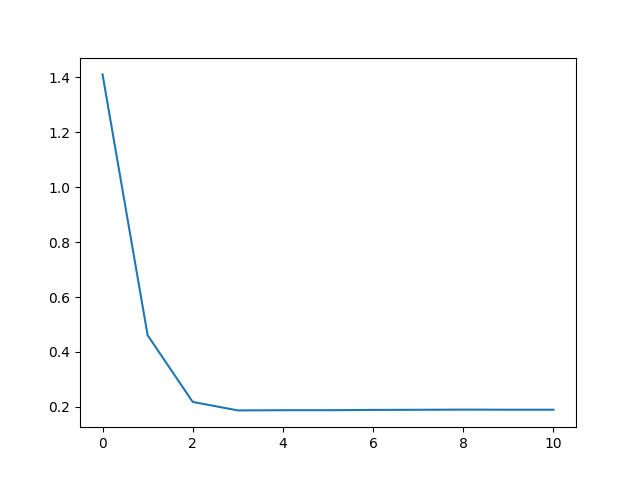
 

Figure 2: PSO clustering

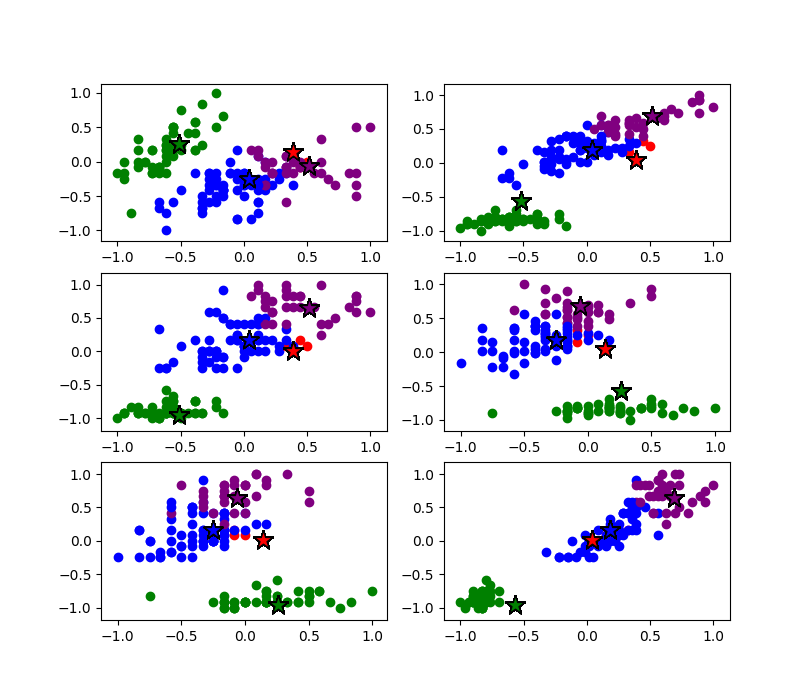
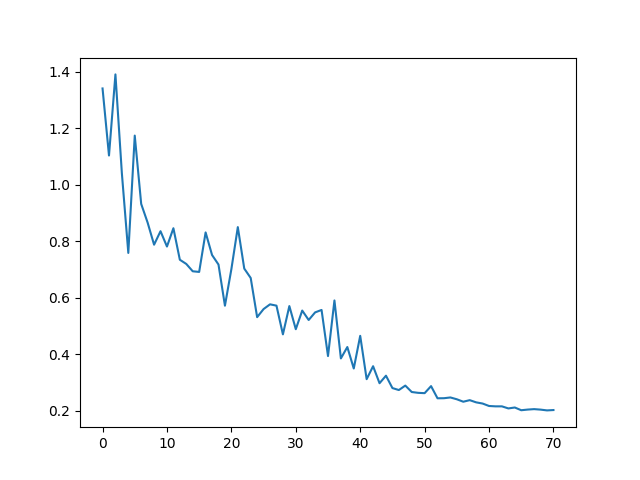
 

Figure 3: Adaptive PSO clustering

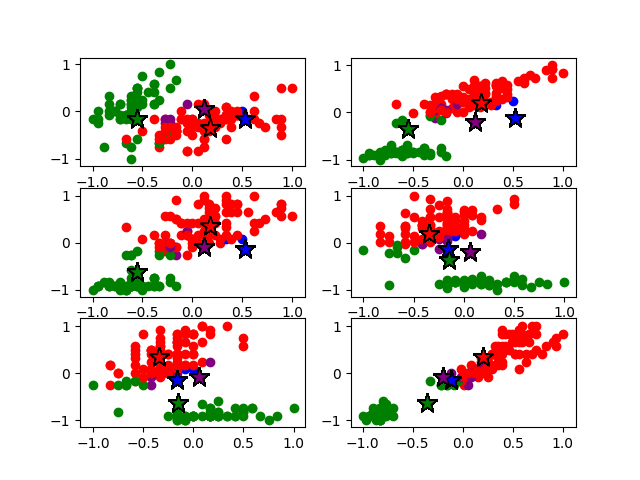
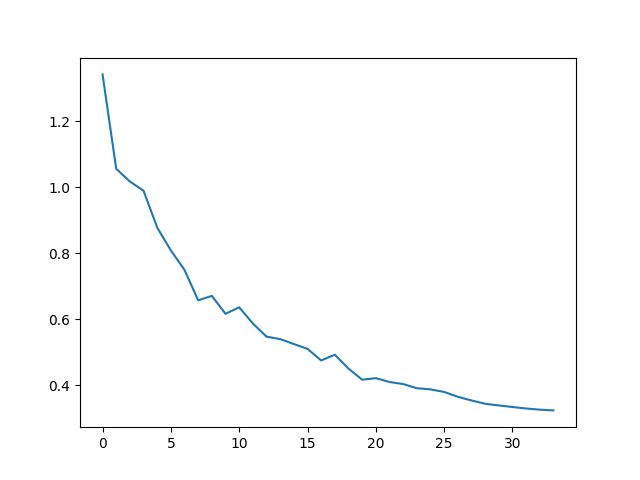
 

Figure 4: Seeded PSO clustering

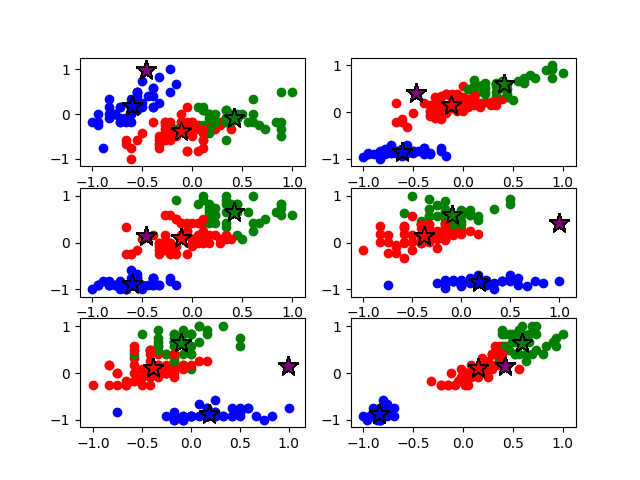
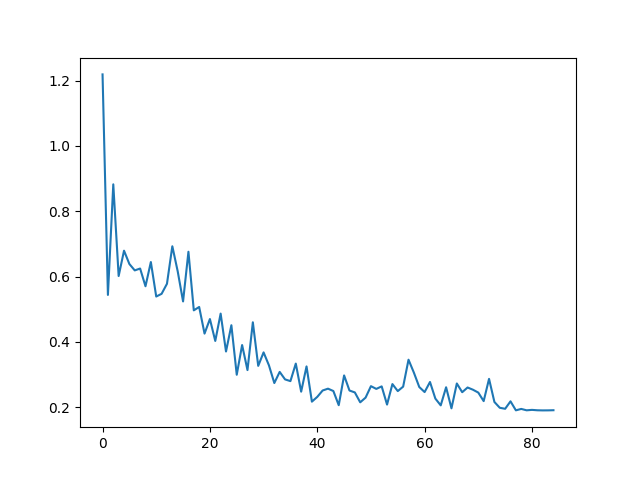
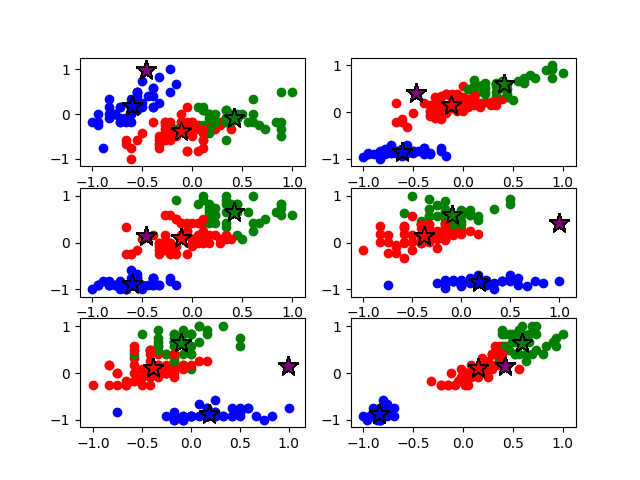
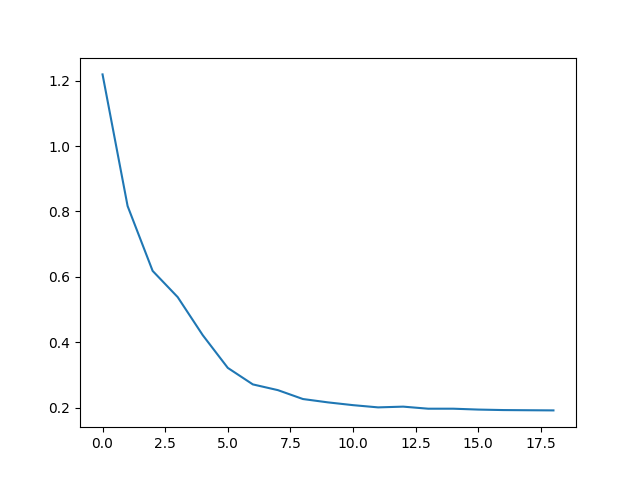
 

Figure 5: Seeded adaptive clustering

# Analysis of Results

In general, the particle swarm clustering and k-means algorithms worked similarly; however, the particle swarm optimization algorithm converged very slowly. This reflects the results found by the Merwe and Engelbrecht paper. While not stated directly, the results of the analysis in this paper show that, while PSO clustering works well for some data sets, it actually yields worse results than k-means for others [1]. The adaptive PSO clustering algorithm performed very poorly on its own. The clustering found was objectively significantly worse than any of the other algorithms. While some of the inter/intra-clustular distance results are comparable to those of the other algorithms, there is a stronger direct correlation between these values in the results for the adaptive algorithm, rather than the desired inverse one. As a result, these somewhat similar results are likely an artifact of the poor clusters creating an asymmetric improvement in these values. In spite of the relatively poor performance of the PSO based clustering algorithms, the seeded variations show better results. As stated earlier, these algorithms are guaranteed to perform at least as well as the k-means clustering algorithm, at least in terms of clustering fitness. These algorithms also provide an improvement to the results of the k-means algorithm, albeit slight. It is important to note, however, that the distances being squared when the values are less than one will make the difference in results appear less significant than they are. the Merwe and Engelbrecht paper, which uses a simple distance metric in its fitness evaluation over a sum of square distances shows a more obvious trend of improvement between the seeded PSO and normal k-means. It can also be seen by the data collected that the seeded adaptive algorithm performs very similarly to the normal seeded PSO clustering algorithm, typically providing a fitness somewhere between the seeded PSO algorithm and k-means. Resultantly, it can be extrapolated that the seeded adaptive algorithm provides some non-negligible increase in clustering capabilities over the normal k-means algorithm. Additionally, while the seeded PSO algorithm typically converges quicker than the non-seeded one, it still converges relatively slowly, whereas the seeded adaptive algorithm converges much quicker. It can also be seen from the graphs depicting the functions fitness to iterations that the adaptive algorithm is much more stable in its convergence.

# Conclusion

While the generic version of the proposed adaptive algorithm performed very poorly in terms of clustering results, when combined with the k-means seeding procedure proposed by the Merwe and Engelbrecht it performs quite well. One of the most likely issues with the algorithms performance is that it reduces the variability of the particle swarm algorithm too much. PSO works by traversing the search space erratically enough to find better solutions that may not have been discovered otherwise. The adaptive approach, however, reduces the variability in the particles search space and attempts to slow them down such that, while the stability is increased, it leads to premature convergence on a suboptimal solution. This essentially removes the random variance created by the acceleration coefficients and random factor, countering the issue that these variables were included to fix [5]. The k-means seeded variation makes this less of an issue by providing a partial solution, and reducing the potential impact of this premature convergence. The algorithm will be able to find better solutions in the immediate vicinity of the k-means cluster, though will be much more likely to underperform when compared to a normal PSO algorithm where the k-means procedure underperforms. Additionally, while the adaptive algorithm increases the convergence rate of the particles over normal PSO clustering and yields decent results when seeded by k-means, it also increases the overhead of the algorithm. Resultantly, this may wash out the effectiveness of such an algorithm in all but vey circumstantial cases.

It is possible that a similar concept could work, though it would have to be modified to maintain more of the random variance in the particles motion. Unfortunately, forcing the particles in a PSO based algorithm to converge quickly has the inherent negative effect of reducing its effectiveness due to the nature of the algorithm; however, it may be possible to utilize a similar technique to improve stability when the particle approaches convergence, allowing slightly better performance, particularly in situations where the particles motion fluctuates unnecessarily near the end of its motion. Reducing the aggressiveness of the adaptive algorithm could also reduce its overhead depending on how this was accomplished.

References

[1] DW van der Merwe, AP Engelbrecht, “Data Clustering Using Particle Swarm Optimization” University of Pretoria, Pretoria, South Africa

[2] A King, “Online K-Means Clustering of Nonstationary Data” 15.097 Predicted Project Report, May 2012.

[3] D.P. Rini, S.M. Shamsuddin, S.S. Yuhaniz, “Particle Swarm Optimization: Technique, System and Challenges”, International Journal of Computer Applications (0975 – 8887) Volume 14– No.1, January 2011.

[4] M.C. Naldi, R.J.G.B. Campello, E.R. Hruschka, et al. “Efficiency Issues of Evolutionary K-Means”, Applied Soft Computing Vol. 11, Issue 2, pg. 1938-1952, March 2011.

[5] J. Kenedy, R Eberhart, “Particle Swarm Optimization” Purdue School of Engineering and Technology, Indianapolis, IN, 1995.

[6] "Particle Swarm Optimization: Tutorial". *Swarmintelligence.org*. N.p., 2017. Web. 27 Apr. 2017.

[7] "15.1. Os — Miscellaneous Operating System Interfaces — Python 2.7.13 Documentation". *Docs.python.org*. N.p., 2017. Web. 27 Apr. 2017.

[8] “Genetic Algorithms Parent Selection". *www.tutorialspoint.com*. N.p., 2017. Web. 27 Apr. 2017.

[9] Lichman, M. (2013). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science

1. [↑](#footnote-ref-1)