An Approach to Distributed Particle Multi-Swarm Optimization

Ben Martin

7839704

COMP 3010

April 16, 2021

[I Abstract 3](#_Toc68168213)

[II Introduction 3](#_Toc68168214)

[III Background 3](#_Toc68168215)

[IV Methods 5](#_Toc68168216)

[1 Client/(Stateless)Server 5](#_Toc68168217)

[2 Client/(Stateful)Server 6](#_Toc68168218)

[3 Peer to Peer 6](#_Toc68168219)

[IV Results/Discussion 7](#_Toc68168220)

[Control 7](#_Toc68168221)

[Client/(Stateless)Server 8](#_Toc68168222)

[Client/(Stateful)Server 8](#_Toc68168223)

[Future Research/Peer to Peer 9](#_Toc68168224)

[Conclusion 10](#_Toc68168225)

# I Abstract

This paper proposes a distributed approach to the well-known Particle Swarm Optimization (PSO) with several modifications. Several variations have been implemented that will be described and tested for evaluation purposes. The evaluation will be based on accuracy, speed, and efficiency. In this context, accuracy describes how close the proposed optimization was to the known global maximum or minimum. In this context, efficiency will be measured by how many times the cost function was evaluated and speed will be measured in iterations. It is natural to assume that with more computers, optimization time will go down, but simply using more hardware to evaluate the cost function more is not as elegant as using more computers to reduce the number of evaluations.

# II Introduction

This paper will implement and evaluate several distributed PSO’s. Each evaluation will be based on several criteria for how the implementation minimizes the given n dimensional cost function. These evaluations will show that given minimal overhead in networking, a distributed multi-swarm optimization algorithm is quicker in converging to an optimal solution.

# III Background

Optimization is fundamental to life [1]. Not to be reductive, but the theory of evolution could be cast as an optimization problem, wherein you take a species and over time change specific characteristics to make it a better fit to the environment. That very notion is the foundation of genetic algorithms. However, the focus of PSO is not in evolution, but in ants.

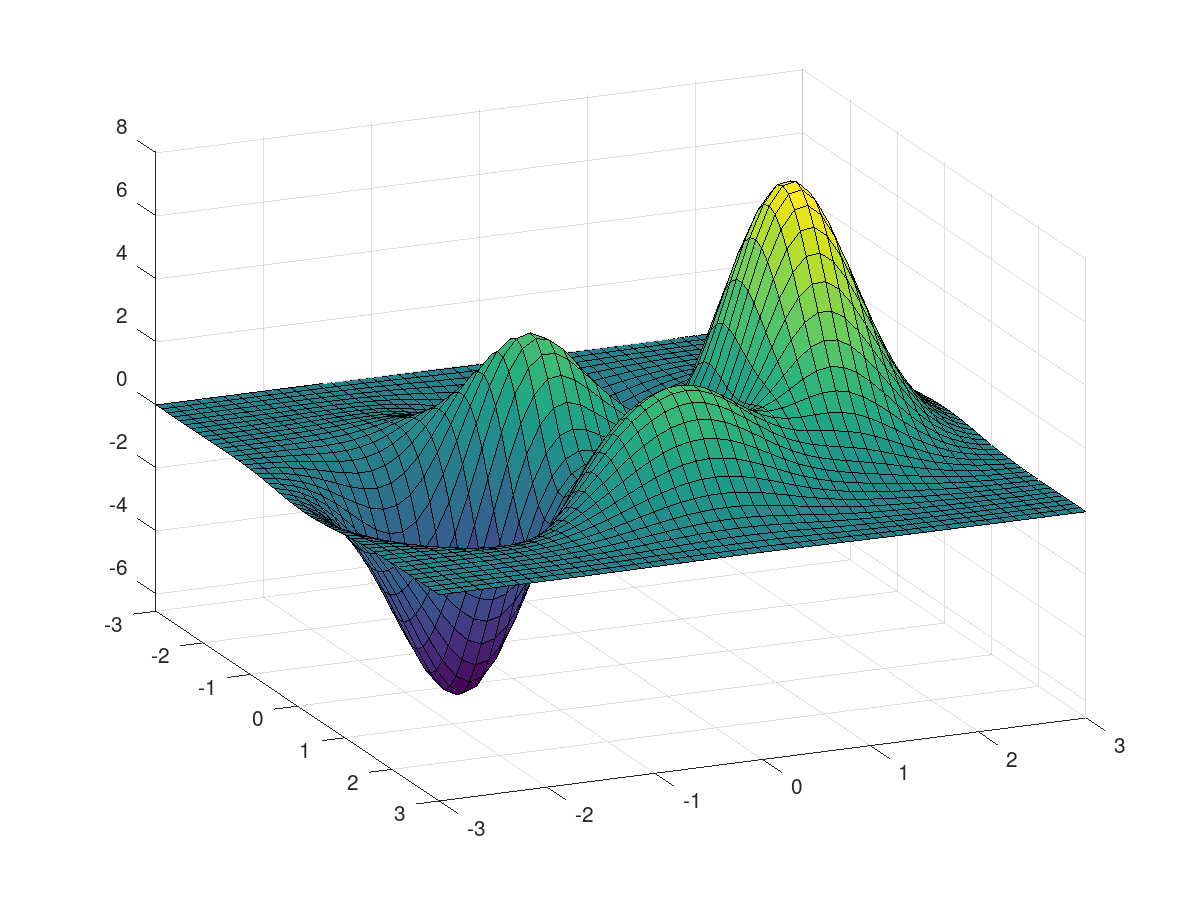


Figure 1. Basic peaks function offered in MATLAB.

Above is just an example of a function with multiple local minima and maxima. To best understand how PSO works, imagine dropping a swarm of ants onto the above domain. Now allow each ant the ability to move freely along the domain and be able to keep track of the minimum position it as an individual has found while also tracking the minimum the swarm has found. With each ant having access to two locations, each ant can ‘pick’ a direction to move in. This direction is referred to as a particle's velocity. Since this method is iterative [1], this velocity will be updated every iteration and will be a function of the local minimum and the swarm minimum. Given enough iterations, the ants will converge into the minimum of -6 while also having explored a large portion of the search space. The result of the optimization will be the swarm minimum, meaning that it is not a requirement for all ants to ‘fall’ into the minimum.

The focus of this paper is not on PSO, but rather a variation of it known as multi-swarm PSO. In terms of ants, imagine having many swarms of ants now, with each having their own swarm minimum that would equate to the specific swarm’s answer to the optimization problem. Meaning, the final answer would just be the minimum of all the swarm minima [2].

# IV Methods

Partitioning and replication is critical to any distributed algorithm [3]. With respect to multi-swarm PSO, partitioning and replication comes quite naturally. The search space for the heuristic becomes implicitly replicated since it is an abstract concept. Additionally, the swarms lend themselves nicely to partitioning where each machine can be responsible for m swarms.

The following will be the methods to be tested. There will also be a control that contains no distribution and will only have a single swarm; however, it is not worth discussing it in its own section.

## 1 Client/(Stateless)Server

As the name suggests this implementation utilizes a client server architecture where the clients will act as workers. The server will not contain any state meaning it will not have the ability to manipulate swarms during optimization. Each client will be given initialization parameters detailing the origin for a particular swarm. As derived from M. Nouri et. al [4], the server could additionally add weights in the velocity equation. Weights in the velocity equation would allow configuring a particular swarm to favour either exploration or convergence. Theoretically this would enable x exploratory swarms with y convergence swarms such that when one of the exploratory swarms finds a new global minimum, the y convergence swarms would converge to that spot allowing the exploratory swarm to continue exploring. This is a nice benefit that distribution offers, but is not possible in the given architecture since the Server is stateless and the clients have no way to communicate their minima to each other.

## 2 Client/(Stateful)Server

This architecture will be identical to the stateless version, but with the ability for the server to hold state; there are now many possibilities for the server to manipulate the swarms during optimization. Dynamic swarm/global minima can now be stored on the server [4]. Doing so would allow the server to add a drift velocity to each swarm enabling the previously mentioned concept of exploratory/convergent swarms. Further, the server will also store the current position of each swarm. The position of a swarm will be defined as an average of each of its particles’ positions. With the positions the server will now be able to add a migratory function [4] to each swarm. When a server detects two or more swarms are within a threshold the drift velocity could be used to migrate the two or more swarms apart from each other. The motivation behind migration is utilizing distributed swarms to increase exploration. If one swarm is exploring a region it would be a waste of resources to allow another swarm to explore any portion of said region.

## 3 Peer to Peer

A potential issue with all methods introduced so far is that they all have a single point of failure. In the case of either server-based method, if the server fails, each client is now running an isolated PSO. Thus, without the server multi-swarm PSO degrades into n instances of the control method. The method introduced in this section will not have this concern, but instead will have other potential issues. The peer-to-peer system will contain n instances of PSO’s with each having the ability to communicate with the others. This method will be capable of the previously mentioned drift velocity and migration functions. For one specific peer to find the global minimum it will need to send a message to n-1 peers with the same being true for finding the position of every swarm. This immediately introduces scaling issues that will be elaborated on in section V. One final consideration is that the peers no longer have a single point of truth, so there needs to be a way to reach consensus on properties that need to remain consistent across peers. For any given problem the only property that must remain consistent is the objective function to minimize. To achieve consensus for the objective function it will be sufficient to have any incoming peer and a veteran peer evaluate their objective function with the same inputs to confirm the functions are equivalent. To bootstrap this process, the first veteran peer will be the node that received the start command from an external source. This will prompt that first veteran node to send assistance commands to all potential workers who will then be considered as incoming peers.

# IV Results/Discussion

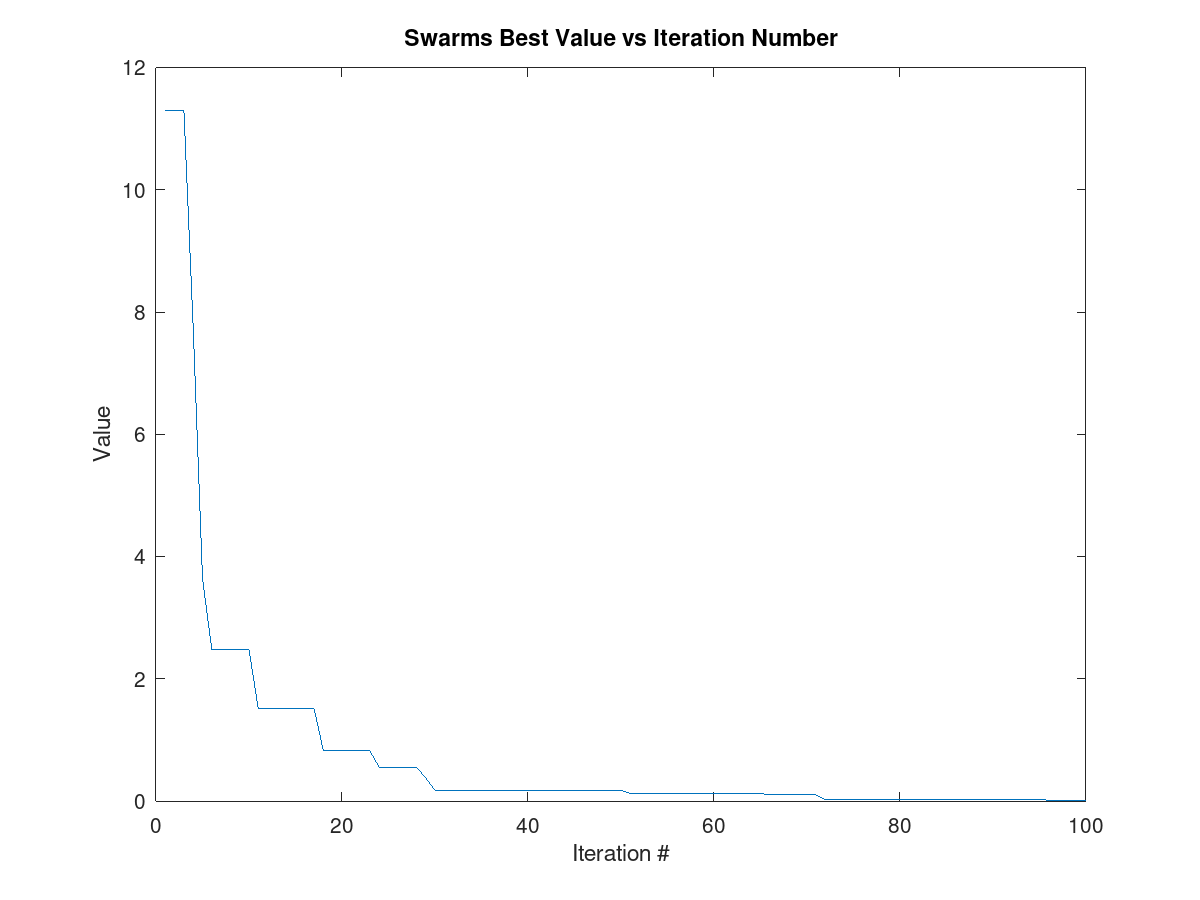
Figure 2 shows the Ackley function which is a common benchmarking function and known for its difficulty [1]. This will be the function used for all following tests on the methods that have been implemented. The global minimum of the Ackley function is 0 at the origin. The function is riddled with local minima making it a challenge to not have a swarm get stuck in one of the minima.

Figure 2. Ackley function example.

### Control

This method is just basic PSO. With a swarm of 100 particles spread out and centered around the coordinates {distributed multi-swarm optimization algorithm is quicker in converging to an optimal solution (9, 9). Figure 3 depicts the convergence rate. After 71 iterations the minimum value found was 0.0106. This method required 7100 function evaluations to arrive at this minimum.

Figure 3. Convergence rate of the control.

### Client/(Stateless)Server

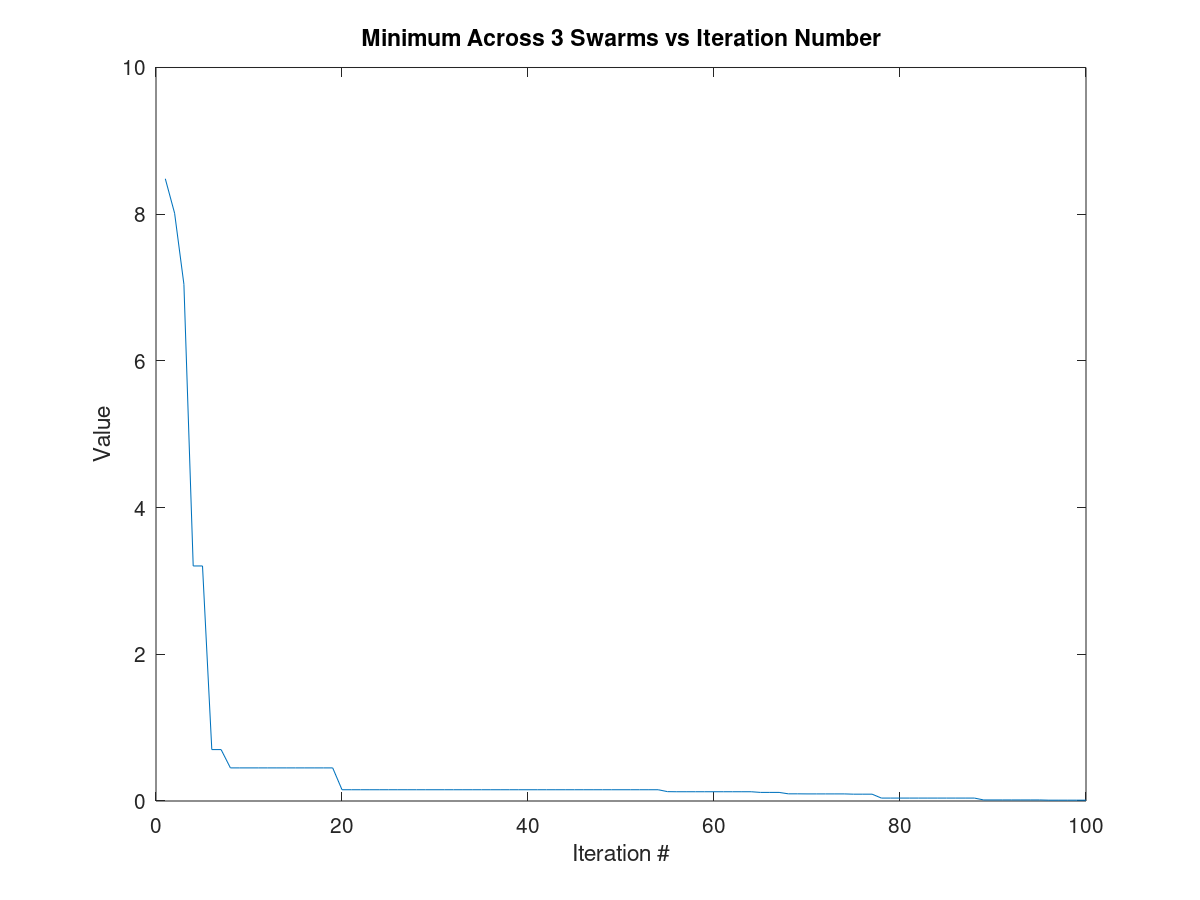
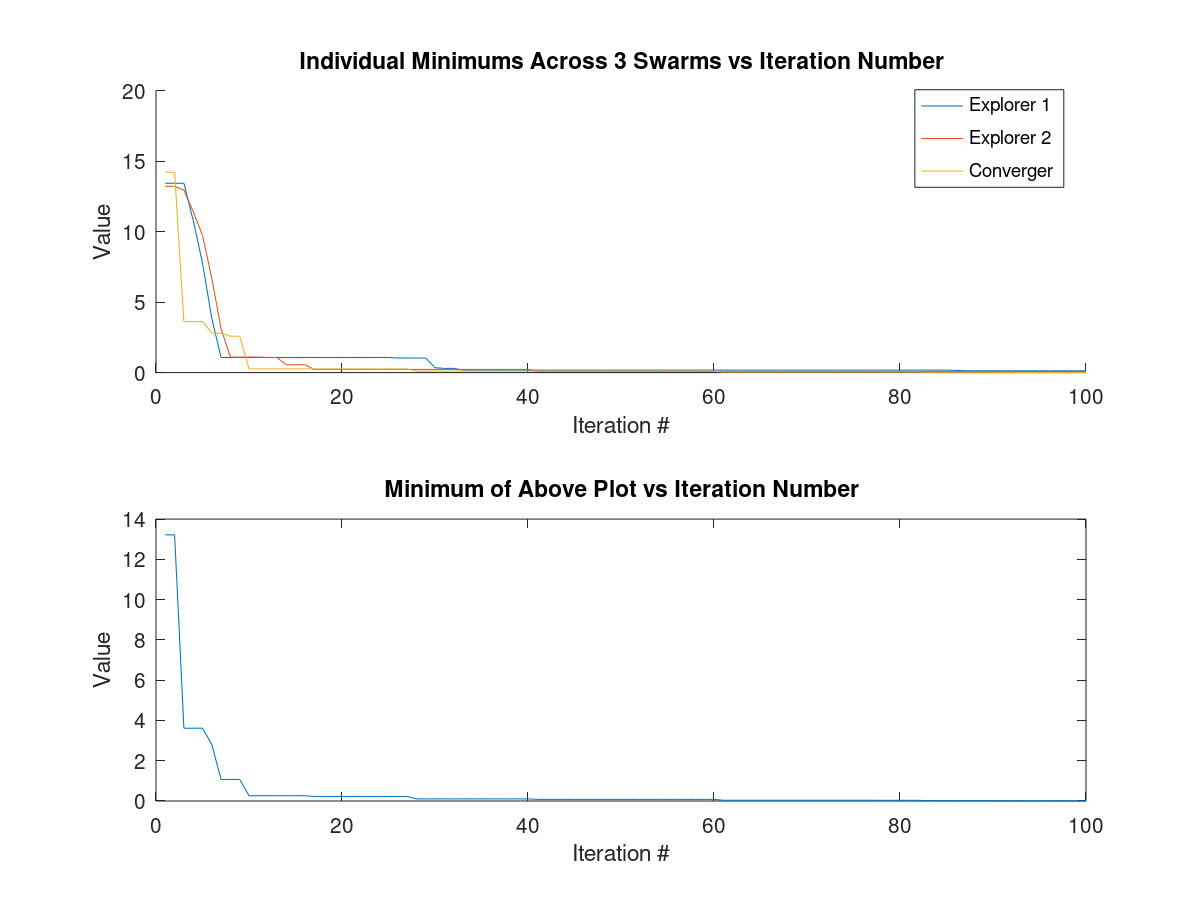
This test used 3 clients each with a swarm of 100 particles. The server positioned the swarms at the coordinates (9, 9) but spread them out immediately. After 78 iterations it converged to a final value of 0.0113. This method used 23 400 function evaluations to arrive at its minimum. As you can see from figure 3 and 4, the Control arrived at a better minimum than the first distributed approach. This is not a surprising result as this approach was little more than just running a basic PSO three times. It is worth noting that the distributed approach did converge faster. The control needed roughly 30 iterations to reach where this method was at 20.

Figure 4. Convergence rate of stateless method.

### Client/(Stateful)Server

This method is like the above method in architecture, but now manipulates the swarms during optimization. Dynamically controlling the swarms is now possible since the clients share state with the server. In particular, the clients share their best-found position and current central position. There are two classes of swarms that were created, the explorer and converger. The explorer is manipulated by the server to ‘bounce’ off all other swarms. The converger is ‘teleported’ to the location of the best solution known across all swarms. After testing this new method under the same conditions, 100 particles per swarm, 100 iterations and a starting point of (9, 9) the minimum found was 0.0068. The number of function evaluations used is again 23 400.



*Figure 5. Convergence rate for individual swarms and the group.*

Figure 5 shows the minimums across all three swarms which highlights the impact that state passing had. As shown, the Converger minimum jumps to the minima found in the explorers’ allowing the converger to continue searching in that section and the explorers to move on. The convergence rate, shown in the second plot, is like the stateless version up to iteration 20; following iteration 20, the stateful method converges much quicker.

### Future Research/Peer to Peer

The stateful method is capable of redirecting swarms to avoid collisions, however, it has no way of redirecting to avoid regions already explored. This results in wasted function evaluations. To be able to avoid regions already explored, the server will simply have to store more state about its clients allowing it to redirect swarms from regions already explored. Increasing the state will result in a scaling issue since the amount of state to store will grow at a rate of O(i\*m) where i is the iterations and m is the number of clients. The peer-to-peer method was not implemented so it will be classified as future research. It is likely the peer-to-peer method would not have converged faster or found a better result since it did not pose any direct benefits to the optimization portion of the system. The major benefits would be in the removal of the single point of failure and natural scaling. The server-based approaches would eventually reach a point where it simply could not handle any more clients whereas the peer-to-peer approach can scale up to what the network will allow. The major drawback of the peer-to-peer method is that the network usage will scale at quadratic rates. Further, the issue of consensus will become much more challenging when attempting to have explorer swarms avoid regions already explored. There is no longer one point of truth leaving the question, which peer should a swarm contact for all regions explored? One potential solution would be having every peer store its history and have each swarm contact all peers every iteration for their respective histories. This will generate a vast amount of network traffic and will likely be a bottleneck when scaling.

# Conclusion

The methods introduced and implemented above have shown that a distributed approach is in fact beneficial to PSO. Logically, increasing the resources would increase the performance which is what the stateless server showed. The stateless implementation was focused on ramping up resources. Increasing resources should of course increase convergence rates since the stateless server was performing 3 times the function evaluations per iteration than that of the control. The true advantage from distribution though came from the stateful method. When comparing the stateful method to the stateless we saw a further increase in convergence rate and a much more optimal solution. Both methods used the same number of function evaluations meaning this method was more than just the simple sum of its parts. From this we can conclude a distributed multi-swarm PSO not only finds a more optimal solution, but is quicker in converging to a solution when compared to its non-distributed counterparts.

References

[1] M. Clerc, Particle Swarm Optimization. Hoboken, NJ: John Wiley and Sons, 2010.

[2] L. Srivastava and H. Singh, "Hybrid multi-swarm particle swarm optimisation based

multi-objective reactive power dispatch," IET Generation, Transmission & Distribution,

vol 9, no 8, pp 727–739, 2015. Available: https://doi.org/10.1049/iet-gtd.2014.0469 [Accessed: 30-Mar-2021].

[3] Takada, M. Distributed Systems for Fun and Profit. [Online]. Available: http://book.mixu.net/distsys/intro.html [Accessed: 30-Mar-2021].

[4] M. Nouiri, A. Bekrar, A. Jemai, S. Niar and A. Ammari, "An effective and distributed

particle swarm optimization algorithm for flexible job-shop scheduling problem," Journal

of Intelligent Manufacturing, vol 29, no 3, pp 603–615, 2018. Available: https://doi.org/10.1007/s10845-015-1039-3 [Accessed: 30-Mar-2021].

Reference Video

<https://youtu.be/boUrIMgqtk4>

Source Code

<https://github.com/BenMartin94/MultiSwarmOp>