Cooperative PSO with Spatially Meaningful Neighbors

Peter Wilson

Supervised by Beatrice M. Ombuki-Berman

Submitted in partial fulfillment
of the requirements for COSC 4F90

Department of Computer Science

Brock University

St. Catharines, Ontario

Abstract

Particle Swarm Optimization (PSO) is a commonly used population-based Computational Intelligence algorithm to optimize continuous functions. [1] It uses communication among 'Particles' to discover optimal solutions to optimization problems. In most cases, the way this communication works is through naive and static connections. A way to improve on this would be to utilize the dynamic distances of the particles to dictate how to communicate and work with particles that are positionally close. However, such a technique would require intensive calculations to determine these distances and is therefore not viable for high dimension problems. This paper proposes and tests an alternative version of this technique, using cooperative swarms to divide up high dimension problems, making this new form of communication possible for complex problems.

Through tests evaluating the robustness of these functions, both with and without the new communication techniques, this paper was able to show that CPSO allows this new communication type to scale with dimensions and still provide it's optimization benefits for more complex problems.

Contents

List of Tables

List of Figures

1 Introduction

Particle Swarm Optimization is a population-based computational intelligence algorithm based on a concept similar to the flocking pattern of birds.[1] It uses the power of numbers to find an optimal solution to a problem over the course of a number of generations. This is done by randomizing potential solutions (particles) and then continuously changing them based on information already determined by the other solutions. As such, it is important that there is a way for the particles to communicate information amongst themselves. The standard form of communication is a 'follow the leader' formation where the best solution tells the rest of the particles to follow them. As the PSO algorithm matures, different communication methods have been introduced.[5]

One such communication method is to have particles communicate with each other if they are searching the same subspace [2] (pockets of 'search space' that has it's own optimal value that may or may not be the overall best). By more directly working together with close particles, they can more thoroughly search the entirety of that subspace. This is known as using 'Spatially Meaningful Neighbors'. To determine who should work with who, the distance between each particle needs to be calculated. This becomes a problem since calculating the distance between all particles for each iteration can be time consuming, even more so if the problem is greater than 3 dimensions. [6] Sufficiently high dimension problems become significantly more complicated and less viable targets for this method and as such, it cannot be used on problems higher than 3 dimensions.

A way of solving this dimension problem would be to split the larger problem into a number of smaller problems. A variation on the standard PSO algorithm is called Co-operative PSO [7] does exactly that. It breaks down high dimension problems into smaller problems that collaborate to solve the main problem. By using CPSO in conjunction with the 'Spatially Meaningful Neighbors', this paper aims to bring that technique to higher dimension problems while still maintaining the subspace searching benefits it offers.

2 Background

2.1 Particle Swarm Optimization

2.1.1 Overview

Particle Swarm Optimization (PSO) is an iterative population-based computational intelligence algorithm which finds an optimal solution for non-linear continuous problems. Computational Intelligence is a domain of study in which the patterns and behaviours of intelligent agents (such as humans or animals) are used to help solve other types of problems [8]. Particle Swarm Optimization specifically draws from the flight patterns of birds in their search of food as well as the schooling pattern of fish. In both cases, the animals tend to follow each other from a distance which allows each to get a different vantage point. [1] When one of the fish/birds find the best source of food (or the best place to land) they all tend to converge and head toward that point.

2.1.2 Particle

PSO builds off this metaphor by representing the birds or fish as Particles. A Particle has a position, velocity and personal best value. It keeps track of all these values and updates them with each iteration. The position of the particle maps logically to the location of where the bird is on a map. The x,y,z,... values of the position also correlate to the input values in the problem which we are trying to optimize. The more input values (variables) the problem contains, the higher dimensions are needed to represent the problem. The initial position of the particle should be randomly generated to allow the swarm to effectively test the entire search space. If they all begin with similar solutions, it is unlikely that all potential solutions will be tested which leads to a higher possibility of converging on a local minimum as opposed to the best overall value. Velocity represents the direction and speed of the Particle. It determines how you update the position after each iteration. The velocity is updated at the end of each iteration as well, and is influenced by the current velocity, the

personal best value (or the best combination of values found previously by the particle which is maintained by the particle) and the global/neighborhood best (the best value found by all the particles in the swarm). The velocity and position are updated each iteration with the following functions:

2.1.3 Position Update Formula

$$x_i = x_i + v_i$$

Updating the position of the particle is as simple as taking the current position and adding the current velocity to it. For multi-dimensional problems where the particles have multiple dimensions, this addition will need to be done for each dimension. In this case, the i represents the index of the position.

2.1.4 Velocity Update Formula

$$v_i = \alpha v_i + \omega C_1(y_i - x_i) + \lambda C_2(\hat{y}_i - x_i)$$

The velocity update function has a little more to it. The new velocity is determined by a combination of it's current velocity, the distance it is from the particles personal best and the distance it is from the global or neighborhood best. [1] In this case y_i represents the personal best for that index and $\hat{y}_i$ represents the global best. The influence that each has is determined by the inertial weight (α), the cognitive weight (ω) and social weight(λ) respectively. These values typically add up to 1 and determine the percentage of influence they have on the new velocity. Additionally, the formula also makes use of a random component. The values C_1 and C_2 are randomly generated values in the range [0,1] that adds

more randomness and aids in keeping the swarm from converging too quickly.

2.1.5 Swarm

The Swarm is the heart that makes PSO work. It is the combination of a number of particles working in unison to solve the same problem. The Swarm needs to maintain the individual particles but it also needs to store a Global Best value. The global best is the position of the best solution found by any of the particles within the swarm. Each particle will in turn use that value to help influence the change in velocity for each iteration as seen in the velocity update formula above (unless an alternative neighborhood topology is used, see Section 2.3). This transfer of knowledge amongst the particles helps the swarm converge on the optimal solution. With each iteration, the swarm tries a number of different potential solutions and gains information to help it decide the overall best solution. The code for the swarm is relatively straight forward:

Table 1: PSO Pseudo Code

```
Initialize particles randomly
while current iteration < max iterations do
    for each particle j ∈ swarm do
        //Update the personal and global bests
        if f(j.x⃗) > f(j.y⃗) then
            j.(y⃗) ← j.(x⃗)
        end if
        if f(j.y⃗) > f(ŷ) then
            (ŷ) ← j.(y⃗)
        end if
    end for
    for all particles ∈ swarm do
        Update the Velocity
        Update the Position
    end for
end while
```

For every iteration, all the swarm needs to do is update the values of the personal best for each particle (represented by $\vec{y}$) and update the global best if a new best value is found.

[1] This is accomplished by passing the solution set into the fitness function (see below) and getting a value. If you are working on a minimization problem, the smaller of the two will become the personal best, otherwise you will take the larger value. It then needs to iterate through each particle, updating it's velocity and position. It is important to complete the global best update before updating the velocity since the global best has an influence on the new velocity. If the global best changes during an iteration, that new global best needs to be reflected in the velocity update in ALL particles, not just the particles that appear afterwards.

2.1.6 Fitness Function

In order to find the best (or optimal) solution to a problem, we first need a way to evaluate those solutions in relation to the optimization problem. The Fitness Function is responsible for taking the list of inputs that make up the solution and determining how well it fits the problem. It will then return a single value that helps the swarm determine if it is a good solution or a bad solution, ultimately helping it converge on the best value. The Fitness Function itself is the representation of the problem that we are looking to optimize and is how we tailor the Particle Swarm Optimization algorithm to solve different problems.

2.1.7 Parameters

There are a number of different parameters that can be altered to affect the success of the algorithm.[1] Increasing or decreasing these values can lead to different results depending on the problem at hand. Experimentation is often required in order to get the best performance out of the algorithm.[9]

Max Iterations: This determines how many iterations the swarm will go through in to search for the optimal solution. Increasing this value can potentially lead to better results, providing it with more time to converge onto the optimal solution, though

better algorithms should require less iterations to hit this target.

Swarm Size: the size of the swarm determines the number of particles it employs to search for the optimal solution. Increasing this gives the swarm more information for each iteration and allows the swarm to have a higher likely hood of finding the overall best solution

Inertial Weight (α): the Inertial weight determines how much of an influence the current velocity has on future velocities. A large Inertial weight will lead to the particle carrying on in it's current direction allowing it to better search the search space (though potentially heading off towards worse values). A low Inertial weight will allow it to converge towards the best values more quickly but may have a hard time searching the rest of the potential values

Cognitive Weight (ω): the Cognitive weight notes the effect the particles' personal best solution has on its future velocity. Increasing this will keep the particle from venturing too far from this solution and only explore the neighboring space unless it leads to better results.

Social Weight (λ): this is called the social weight because it determines the effect other particles have on the current particle. This is the influence the global best (or the best solution currently found by the entire swarm/neighborhood) has on the future velocity of each particle. Different neighborhood layouts/topologies can change the effect of the social communication (See 2.3). A low value will have the particles mostly work on their own, whereas a large social weight will have all particles converge toward the best value.

2.2 Co-operative Particle Swarm Optimization

Although Particle Swarm Optimization is a very successful algorithm for solving continuous problems with reasonable dimensions, it often struggles as those dimensions increase. If the problem has a large enough number of inputs, it can be difficult for the swarm to hone in on an optimal solution given the breadth of the search space size, often leading it to get stuck in local optima. This issue is often referred to as the "curse of dimensionality" and affects most other stochastic optimization algorithms [7].

Much of the impetus of this problem spawns from having all the dimensions update at the same time and then calculating the fitness of the function. This can result in a situation where a new solution is objectively worse for most dimensions but leads to a better fitness value overall due to a better value for a single dimension.[10] For instance, if the intention of the algorithm is to minimize the sum of the values (represented as $\sum_{n=0}^{d} x_n$) the position [5,5,5] will give a fitness value of 15, while the position [6,1,7] returns a better value of 14 even though the first and last dimension values are moving away from the optimal solution.[10] This can influence the velocity to lead to the particles in the wrong direction, away from the optimal solution.

CPSO attempts to solve the "curse of dimensionality" by instead dividing the problem into multiple sub problems.[7] This allows you to solve a subset of the dimensions separately by converting a single high dimension problem into a number of low dimension problems. With each iteration, the particles are then combined together to get a total fitness value. It is important to not update all these particles before gathering the fitness value as you will lead to similarly counter productive results. Different variants of the algorithm introduce alternative ways of splitting the larger problem. An issue which arises from this separation is that it removes the dependencies of variables. For instance, if two dimensions are required to work together to improve the fitness, they will be no longer able to do so directly if they are being updated with different swarms.

2.2.1 CPSO-S

The original version of CPSO is called CPSO-S. Developed by Van den bergh and Engelbrecht [7] it tackles the problem by dividing an n-dimensional problem into n 1-dimensional swarms. It solves each swarm separately and then merges them to get the fitness value. It does this by creating an initial randomly generated solution and only updating the part that the current swarm is optimizing to test the fitness. It will continue using this randomly generated solution throughout the algorithm to ensure the other swarms do not affect the convergence of the current swarm. The rest of the algorithm is identical to the standard Particle Swarm Optimization, however on top of iterating through each particle per iteration, you also need to go through each swarm.

Table 2: CPSO-S Pseudo Code

```
create and initialize n one-dimensional PSO Swarms
Randomly initialize solution vector
while current iteration < max iterations do
    for each swarm i ∈ swarms do
        for each particle j ∈ i do
            //Update the personal and global bests
            if f(j.x⃗) > f(j.y⃗) then
                j.(y⃗) ← j.(x⃗)
            end if
            if f(j.y⃗) > f(ŷ) then
                (ŷ) ← j.(y⃗)
            end if
            Update the Velocity
            Update the Position
        end for
    end for
end while
```

Since 1 dimensional problems are trivial to solve through Particle Swarm Optimization, separating the problem into 1-dimensional problems significantly simplifies the end goal. However, since each swarm operates almost in a vacuum, it has no idea of the values obtained by the other problems. This means if one value is dependent or affected by another, there

is no way of that being taken into account. This leads to the swarms getting stuck in a position that is locally optimal for each dimension but is not the global best solution to the problem.

2.2.2 Variant: CPSO-S$_k$

The first variant proposed by Van den bergh and Engelbrecht is CPSO-S$_k$[7]. Instead of dividing the problem dimensions into n 1-dimensional swarms, it now evenly divides the problem into k swarms (or as close to even as possible). The basis of this variant is to attempt to combine dimensions that are dependent on each other in hopes of overcoming local optima. The rest of the algorithm is identical to CPSO-S. It optimizes those multi-dimensional swarms separately and has them converge on a local optima, ultimately combining the individual solutions as it's final optimization value.

Table 3: CPSO-S$_k$ Pseudo Code

$K_1 \leftarrow nmodK$
$K_2 \leftarrow K - (nmodK)$
Initialize $K_1\lceil n/K \rceil$-dimensional PSOs
Initialize $K_2\lfloor n/K \rfloor$-dimensional PSOs
Randomly initialize solution vector
while current iteration < max iterations **do**
  for each swarm $i \in swarms$ **do**
    for each particle $j \in i$ **do**
      //Update the personal and global bests
      if $f(j.\vec{x}) > f(j.\vec{y})$ **then**
        $j.\vec{(y)} \leftarrow j.\vec{(x)}$
      end if
      if $f(j.\vec{y}) > f(\hat{y})$ **then**
        $\hat{(y)} \leftarrow j.\vec{(y)}$
      end if
      Update the Velocity
      Update the Position
    end for
  end for
end while

However, though this leads to a better chance of matching dependent dimensions, it still tends to get stuck in local optima. This is because the dimension split that exists in the CPSO solutions don't exist in the actual problem. By dividing the problem into sub problems, we are fundamentally altering the initial problem. It is also unlikely that dividing the algorithm into even sections will capture all the connected values and since there is no communication between swarms, it is impossible for the algorithm to break out of those local optima.

2.2.3 Variant: CPSO-R_k

One potential solution to the local optima problem would be to randomly divide the problem into differently sized sub problems. This leads to a larger likelihood of combining dependant variables though that is entirely left up to chance. If it does successfully make those connections it can very quickly converge on the optimal solution. CPSO-R_k (also known as Random CPSO) tackles this methodology by dividing up the problem into k randomly sized PSOs. Once the swarms are created, it functions very similar to CPSO-S_k. It solves all the swarms separately, placing them into a temporary solution vector to get the fitness. Once the max number of iterations is reached, it will take the global best of each swarm and combine them into a single solution.

2.3 Neighborhood Topologies

The Neighborhood Topology of a PSO dictates how the Swarms' individual particles are allowed to communicate with each other. This communication is the real strength of the PSO as it allows its particles to work together in solving the problem. This particle communication is achieved through the use of 'Global Best' values. For each particle, when it is updating it's velocity for the next iteration, it will consult the global best in its neighborhood to aid in determining it's next move. In the standard PSO (Section 2.1.5), all particles are able to communicate with each other. This means that all particles in the swarm are 'led' by

the overall best particle in the entire swarm. This behaviour can sometimes lead to a naive swarm that quickly converges on an 'optimal' solution without fully exploring the entire search space of the problem. To combat this, a local PSO [3] was proposed, which utilizes different, more constrained, neighborhood topologies in hopes of creating a better PSO .

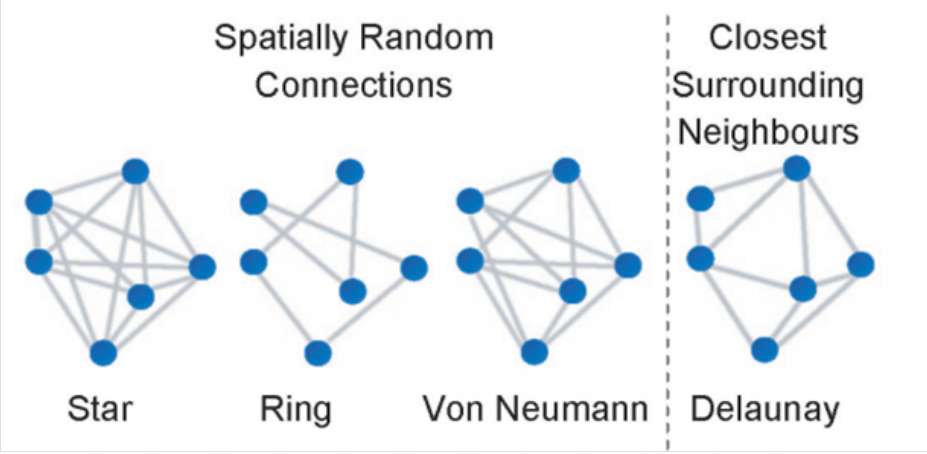

Figure 1: Visual Representation of the Neighborhoods[2]

2.3.1 Common Spatially Random Topologies

There are a number of Neighborhood Topologies that are currently being used and tested in the PSO community. Many of these topologies have their own pros and cons and can lead to different or better results depending on the problem in question. These topologies are known as 'Spatially Random' since the location of the particles at any point in the calculation has no bearing on what particles it is connected to. [2]

Star

The Star Topology is the most common since it is used in the standard PSO. As mentioned, Star Topology connects every particle with each other. That means the 'neighborhood best' for every particle is the same as the 'global best' position in the swarm. The Star Topology uses this information to quickly converge onto the optimal best without a thorough exploration of the search space.

Ring

The Ring Topology changes the neighborhood topology by reducing the connections per particle to just two. All the particles are ultimately still connected to each other, but through multiple intermediary particles. This topology slows the transfer of information among the particles, decreasing the impact the global best has on the swarm itself[3]. This helps to slow down the convergence of the swarm, allowing for a more thorough investigation of the search space. However, as a result, it requires a much larger number of iterations to find an optimal solution. Additionally, the connections are often determined randomly during the initialization of the particles. As a result, the effect of the topology can be dependant on the success of that initial setup.

Von Neumann

The Von Neumann Topology expands on the Ring Topology by increasing the number of connections each particle has while still maintaining the slow information transfer rate within the network[11]. This helps to speed up convergence while allowing for exploration. The particles are connected in a cubic-lattice arrangement [2] where particles are connected to particles on the left, right, above and below. This isn't, however, determined by actual location of the particle, but often based on the index value that particle is in the Swarm array, where above and below are determined by some sort of offset based on the swarm size (See Figure 2). This means it does not take advantage of positioning information and the connections are not updated throughout the completion of the run.

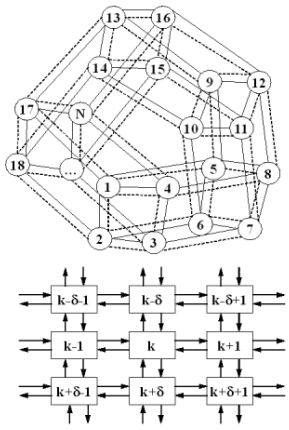

Figure 2: Von Neumann[3]

2.3.2 Spatially Meaningful Neighbors

Where the updates to the Spatially Random Neighborhood Topologies allowed for more time in searching the search space, the actual searching ultimately ends up being mostly random. This is due to the fact that these connections are both randomly selected and only determined once during the life cycle of the swarm. If the connections were dynamically chosen based on information within the swarm, that information could be used to better direct the collaboration between particles. In the paper "Particle Swarm Using Spatially Meaningful Neighbors" [2] Lane, Engelbrecht and Gain proposed using the distance between particles to determine which particle is connected to which. It was theorized that particles that are close together have a high probability to be searching the same subspace (a pocket of space that has it's own local max/min). The proposed new Neighborhood Topology called 'Spatially Meaningful Neighbors' uses the Delaunay Triangulation algorithm to connect all particles with the neighbors that are closest to them. Those connections then represent which particles they can talk to and draw information from. Decisions are then made to avoid or proceed in searching a space based on how the particles are moving (See Section 2.4). This led to a more guided search and a higher success rate in their tests. [2]

2.4 Dynamic Connections

Just having the list of the closest neighbors for each particle isn't in itself inherently helpful for Particle Swarm Optimization. Of this list, we now want to determine which of these particles are are 'working together' so we can have the particles cooperate to find a local optima.[2] Setting up connections between these particles will allow them to communicate and share information, allowing them to more quickly converge on that optima. A particle is considered to be working with another if:

- Particle P_1 is following behind Particle P_2. i.e. both are heading in the same direction. This this case, we'd want to establish a directed connection so the particle in front can send information to the trailing particle[2]
- Both Particle P_1 and Particle P_2 are heading towards each other but not passing each other. Here we would want to set up an undirected connection so that both particles can share information.[2]

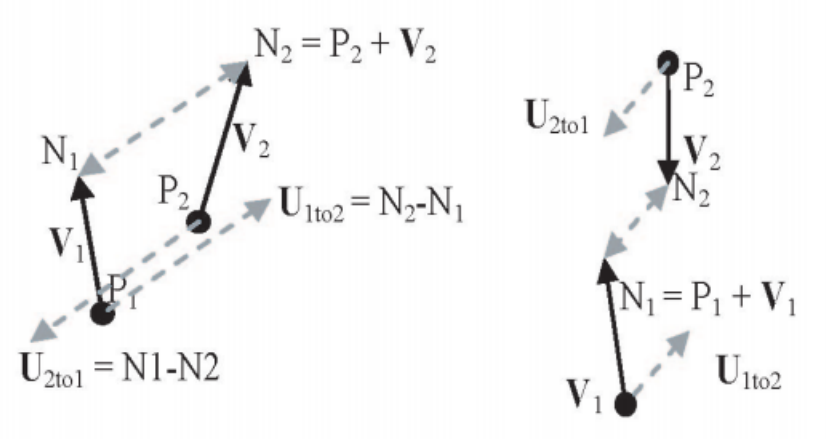

Figure 3: How to know if the particles are working together[2]

You can test both cases with a simple equation:

$$V_1 \cdot U_{1to2} > 0$$

In the function above, the V_1 represents the current velocity of P_1 and U_{1to2} is the vector representing the distance from P_1 to P_2 after the two Particles' next position update. Using the dot product on both these vectors allows us to determine the angle between them. If the dot product is greater than 0, we know the angle between the two vectors is less than 90 degrees. This outcome can only occur if it satisfies one of the two.

A caveat to this would be that we still wish to encourage local exploration. Delaunay Triangulation does not guarantee that the particles are close together. If two neighboring Particles are working together, but are still reasonably far away, having them communicate will inhibit local exploration. Therefore we need to set a threshold in which we want the particles to search within before they start communicating. Table 4 details the process of finding the best neighbor to communicate with.

Table 4: Choosing The Best Neighbor

input: N_i Particle i's closest neighbors
procedure CHOOSEBESTNEIGHBOR(N_i)
    hasConnectedNeighbors = false
    $P_f \leftarrow min(N_k)$
    for $k \leftarrow 1toneighborset_size$ **do**
        if $working_together(P_i, P_k)$ and $dist(X_i - P_c) < dist(X_i - P_k)$ and $f(P_k) < f(X_i)$ **then**
            $P_c \leftarrow P_k$
            hasConnectedNeighbors $\leftarrow$ true
        end if
    end for
    $localExploitationRation \leftarrow swarm.diameter/200$
    if $hasConnectedNeighbors$ and $distance(X_i - P - c)/distance(X_i - P - f) > localExploitationRatio$ and $V_i < 2 * distance(X_i - P_x)$ **then**
        $P_n \leftarrow P_c$
    else
        $P_n \leftarrow P_f$
    end if
return P_n
end procedure

2.5 Delaunay Triangulation

2.5.1 Triangulation

Triangulation in Computational Geometry is the decomposition of a polygon or set of points into a set of pairwise non-intersecting triangles in which all points in the set or edge in the polygon are an edge in at least one of newly the formed triangles.[2] In n-dimensions, the triangles are instead represented as simplices (the n-dimensional equivalent)[12]. Triangulation is mainly used in computer graphics to simplify complex shapes into easily drawn triangles, though the information gathered from this process can be utilized in a number of different problems. An optimal triangulation is one that minimizes the total length of all the edges, otherwise known has having the minimum weight. What this means is, an optimal triangulation will, on average, connect all the points to their nearest neighbors. Definitively finding the optimal triangulation of a set of points is considered to be an NP-Hard problem[8], however, there are a number of triangulation algorithms that do a very good job of coming close to optimal.

2.5.2 Delaunay Triangulation

Delaunay Triangulation (DT) is considered to be one of the most efficient triangulation methods in computational geometry and often leads to optimal or close to optimal triangulations.[8] This triangulation methodology creates a set of triangles such that the circumcircle of each triangle contains no other points. What this means is that if one were to draw a circle around every triangle where each point of the triangle lies on the circumference of that circle, no points will fall inside another triangles circle.[2] This guarantees the the triangles are evenly spaced out and often results in a lower number of 'thin' triangles.

The issue with Delaunay Triangulation is one of dimensionality. In 2-dimensions, Delaunay Triangulation can be done with a worst-case time complexity of just O($nlogn$) which is one of the quickest methods for pseudo-optimal triangulation[2]. However, it becomes

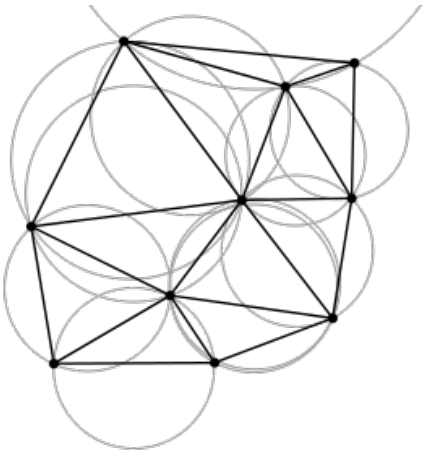

Figure 4: Circumcircles drawn amongst the Triangles[4]

increasingly difficult to compute as dimensions increase making it a less viable solution to all versions of problem. The majority of algorithms that compute Delaunay Triangulation are typically limited to just 2 and 3 dimensions. One way of computing DT in higher dimensions would be to increase the dimension by 1 (giving the value of the new dimension as $|p|^2$) and then computing the convex hull on those sets of points [8]. However, as the dimensions increase even that becomes increasingly complex to compute. In 4-dimensions the best case algorithm will have a worst-case $O(n^3)$ run time and in d-dimensions, it takes $O(n^{\lceil d/2 \rceil + 1})$ time.

3 Literature Review/Previous work

3.1 CPSO

In van den Bergh and Engelbrecht's 2004 paper "A Cooperative Approach to Particle Swarm Optimization"[7], they introduced a new concept called Cooperative Particle Swarm Optimization (CPSO). They noted that, like other stochastic optimization algorithms, PSO suffered from the "curse of dimensionality". This references the increasing difficulty of finding an optimal solution as the dimension increases since the potential search space increases exponentially. Taking note of Potter's dimensional decomposition to help solve this problem

in Genetic Algorithms (another stochastic optimization algorithm)[13], they attempted to do the same for Particle Swarm Optimization and in the process, solving the issue of pseudominima, in which each swarm gets stuck in a local optima which isn't globally optimal.

Their approach was to create two new PSO algorithm variants. The first was CPSO-S_k, which took the idea of dividing the problem into smaller dimensions, but instead of n 1-dimensional sub problems, it is divided into k problems of equal size. This is in direct response to the issue of pseudominima, which spawns from the issue of each swarm optimizing completely separately. The division is now done in hopes that the dependent variables are grouped together in the same smaller sub swarms. The second proposed variant was CPSO-H_k (Hybrid CPSO). This took advantage of PSOs ability to overcome local optima and CPSO's ability for quick convergence by having both CPSO-S_k and a standard PSO run simultaneously. The two algorithms would share information and help each other find a global optima

Their tests revealed that both variants of CPSO vastly outperformed the standard Particle Swarm Optimization algorithm in higher dimension problems. Each algorithm was tested against a number of standard optimization formulas with 30 dimensions each and not only did they see an increase in terms of the quality of solutions returned, but also in the robustness of the solutions (meaning they were able to reach an optimum solution prior to hitting the max loops). CPSO algorithms were able to converge onto solutions significantly faster than the standard PSO while still leading to global optimums. The problem of pseudominima still persisted in the CPSO-S_k, though at a lesser degree, however the other proposed variant CPSO-H_k performed much better in this case. While the Hybrid variant required more memory space and processing power to compute, it was able to overcome local minima better than its counterpart and led to the best solutions.[7]

Overall, they found that CPSO is an essential tool for solving optimization problems in higher dimensions and leads to objectively better results than it's non-cooperative counterparts

3.2 Delaunay Triangulation

Particle Swarm Optimization is a powerful algorithm for optimizing continuous non-linear function by having Particles swarm around the search space to help converge on the global optimum value. One way PSO can be improved upon is by introducing communication techniques to allow the swarms to more directly work together and 'team up' to better evaluate a local search space. A number of Particle knowledge-transfer topologies have been introduced in the past, however, in Lane, Engelbrecht and Gain's paper titled "Particle Swarm Optimization with Spatially Meaningful Neighbors"[2] They propose a different topology utilizing Delaunay Triangulation's ability to easily locate each particles closest neighbor. In this paper, they propose the idea that trading information amongst close particles allow for better use of the local search space.

Delaunay Triangulation is a quick and effective way of finding neighbors in low dimensions that typically represent the closest neighbors to each particle. The algorithm returns triangles which make use of all the particles and also has approximately the minimum-weight (or shortest overall edge lengths). Lane, Engelbrecht and Gain use this information along with a number of other heuristics to determine which particle should communicate with which for best result. They noted that having particles that are 'working together', share information allowed them to collaborate to better explore the smaller local area.[2] Particles were defined as working together if the were headed either in the same direction or towards each other. They also noted that communication with the optimal neighbor was also effective as long as the neighbor wasn't too far way.

After testing their theory, they were able to conclude that Delaunay Triangulation for information sharing between particles led to much better results in low dimensions.[2] Its success rates were almost uni-formally higher than the other knowledge transfer approaches, meaning it was more likely to find the global optimum solution. The problem arose with larger dimension problems. As the dimension rose, the Delaunay Triangulation increasingly became the algorithms bottleneck. In 2 or 3 dimensions, Delaunay Triangulation is a quick

way of determining the particle's nearest neighbors. In larger dimensions, however, it becomes significantly harder to compute and is therefore less helpful for use in Particle Swarm Optimization[8]

Lane, Engelbrecht and Gain go on to conclude that the use of spatially significant neighbors to transfer knowledge is comparatively more successful to the other common knowledge transfer topologies. The use of Delaunay Triangulation, however limits the approach to just 2 and 3 dimensions.[2]

4 CPSO using Delaunay Triangulation

Building off of the previous work, it is clear the major downside to the use of Delaunay Triangulation spawns from its issues in higher dimensions. In the past, such issues were solved by utilizing CPSO's ability to break the problem into a number of smaller sub problems, as seen with PSO's "curse of dimensionality"[10]. Knowing that it has been known to lead to better results, it would be worth trying Delaunay Triangulation in conjunction with CPSO algorithms to see if they can both benefit from each others skills to better solve problems in n-dimensions. It was stated that the tipping point for this benefit is for problems greater than 3 dimensions, as such, all CPSO variations have been initialized such that no one swarm is handling more than 3 dimensions.

4.1 Experimental Setup

To test this theory, a number of known CPSO algorithms were implemented with the added functionality of the communication via spatially significant neighbors using the Delaunay Triangulation approach. The following algorithms were tested for this experiment:

- **PSO**: For comparison's sake, all CPSO variants will be compared to the standard PSO algorithm to ensure either of the changes lead to a meaningful benefit. All the values will be identical to their CPSO counterparts.

- **CPSO-S**: This variant restricts each swarm to just one dimension. It solves all dimensions dependently of each other. In problems of a single dimension, the distances are easily calculated directly instead of using the Delaunay Triangulation algorithm. This is due to Delaunay Triangulation not working for problems under 2 dimensions.

- **CPSO-S$_k$**: Due to Delaunay Triangulation being limited to just 2 and 3 dimensions, a k needed to be selected in which the swarm is divided into 2 or 3 dimension chunks. For the purposes of this test, swarms were divided as such that each swarm solves 2 dimensions. ($k = \frac{dimensions}{2}$)

- **CPSO-R$_k$**: The Algorithm was altered to ensure all the random swarms are selected to only be between 1 and 3 dimensions to ensure the spatially significant neighbors algorithm works. The number of swarms is set to half that of the overall dimensions to allow for an even distribution of 1 dimension and 3 dimensional swarms.

4.2 CPSO configuration

The CPSO and PSO parameters were selected to match those tested in both the Delaunay Triangulation and the CPSO paper in attempt to replicate the results achieved in those experiments. As per the other experiments, the number of iterations were capped at 10,000[7]. To keep the playing field even, the particles allotted to each algorithm will be distributed evenly amongst each swarm. This means that each iteration for the swarm will contain the exact same number of fitness calls, allowing direct comparison of the results. Each experiment was performed 50 times; the reported results are the average of the global bests after all of the 50 runs. The experiments were also repeated for each type of swarm using 15, 20, and 25 particles for 6 dimension problems and 50,75,100 for 20 dimension problem with and without the use of spatially significant neighbors. All variants of CPSO used the following input values:

- **Social Weight (λ)**: 1.49

- **Cognitive Weight** (ω): 1.49
- **Inertial Weight** (α): starts at 1 and linearly decreases with each iteration, ultimately hitting 0 on the last iteration
- **Max Velocity** (V_{max}): clamped to the domain of the function
- **Max Position** (P_{max}): also clamped to the domain of the function
- **Particle Count**: particles are evenly distributed among the swarms in the algorithm. for example, if the particle count is set to 100 and the CPSO has 10 different swarms, each swarm will only have 10 particles to solve its particular sub problem.

4.3 Benchmark functions

Again, in an attempt to replicate the results of the previous experiments, for comparisons sake, the same benchmark functions were selected here as were in the two previous papers. The functions which were selected are as follows:

Function	Domain	Criterion
Rastrigin	[-5.12;5.12]	0.01
Rosenbrock	[-30;30]	100
Griewanck	[-600;600]	0.05
Ackley	[-32;32]	0.01

Table 5: Functions, Stop Criteria and Domains

The "Success Rate" evaluation indicates the percentage of runs that reached the end criterion and the "Iterations to Criterion" value returns the average number of iterations it took to reach that stopping point.

The functions themselves are as follows:

Rastrigin:

$$\sum_{i=1}^{n} x_i^2 + 10 - 10cos(2\pi x_i)$$

Rosenbrock:

$$\sum_{i=1}^{n-1} 100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2$$

Griewanck:

$$\frac{1}{4000}\sum_{i=1}^{n} x_i^2 - \prod_{i=1}^{n} cos(\frac{x_i}{\sqrt{i}})x_i + 1$$

Ackley:

$$20 + e - 20e^{-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}}} - e^{\frac{\sqrt{\sum_{i=1}^{n} cos(2\pi x_i)}}{n}}$$

4.4 Experiment 1: Robustness Test on Lower Dimensions

The mark of a good PSO algorithm is one that can hit or get very close to the optimal result on a consistent basis. As such, it is important that the robustness of the algorithm is tested against other similar algorithms to see how it compares. In this case, "Robustness" refers to the percentage of successful runs and the number of iterations required to reach that success over the course of 50 runs. A successful run is one that reaches the threshold for the optimal solution before the maximum number of iterations is reached. The thresholds vary per function (see 4.3) but the max number of iterations is set to a constant 10,000 iterations for all tested algorithms. To make the iterations equivalent for all the various types of PSO algorithms, the total number of particles per algorithm type is divided evenly amongst all available swarms. This means each iteration evaluates the same number of particles regardless of swarm count. The resulting tables are structured as such that the algorithm being tested is in the first column, followed by the number of particles used in that calculation. Next is the percentage of successful runs with the average iterations for tests using the Star Topology followed by tests using the Spatially Meaningful Neighbors Topology. The best runs are those that have the highest success percentage while also taking the least number of iterations to achieve it. It is to be noted that if the run was not successful, it's iteration count does not factor into the average iterations for the test. The Spatially Meaningful Neighbors Topology was not tested for PSO since it has already been

determined that the method shouldn't be used for swarms larger than 3 dimensions. For this first experiment, tests were run against problems of 6-dimensions to see if the Delaunay Triangulation calculations led to noticeable improvements.

4.4.1 Results

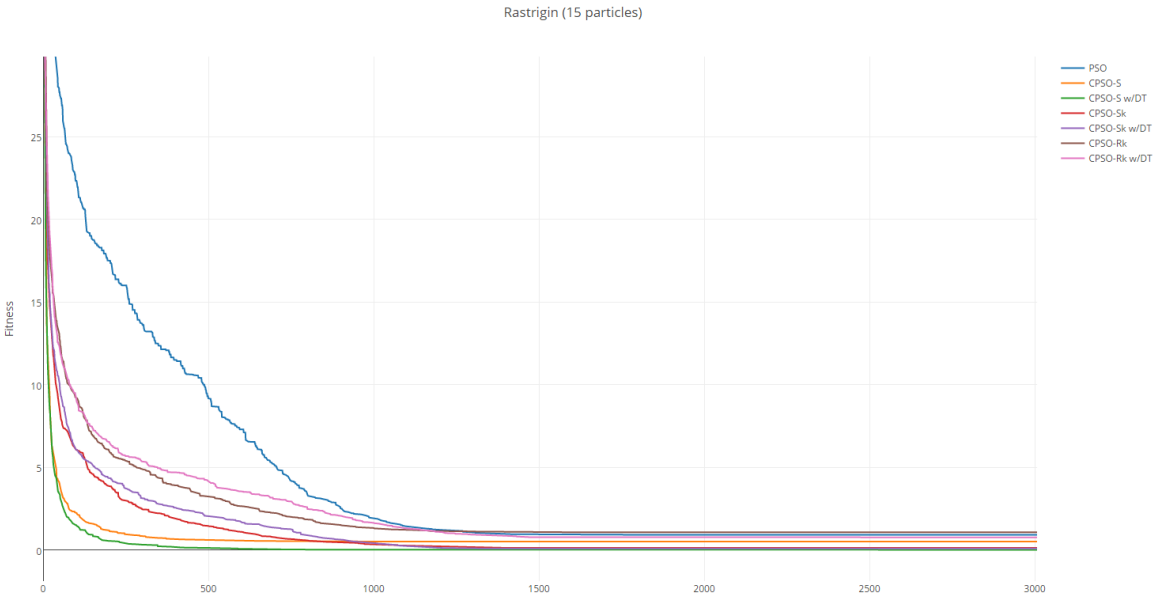

Figure 5: Rastrigin Comparison (n = 6, particles = 15)

Algorithm	s	Standard		Spatially Meaningful Neighbors	
		Succeeded	Iterations	Succeeded	Iterations
PSO	15	54%	1291		
	20	63%	1264		
	25	60%	1185		
CPSO-S	15	100%	837	100%	772
	20	100%	597	100%	624
	25	100%	462	100%	465
CPSO-S_2	15	90%	1091	92%	1191
	20	98%	1033	100%	1165
	25	98%	969	100%	1145
CPSO-R_3	15	58%	1173	74%	1295
	20	80%	1110	80%	1312
	25	88%	1032	84%	1314

Table 6: Rastrigin Robustness Analysis (n=6)

Table 6 details the results of the Rastrigin function against all tested algorithms over the course of 50 runs. Of all 4 algorithms tested, the standard PSO has the lowest success rate as well as the highest average iteration count for the runs that were successful. The CPSO-S algorithm was able to converge on the success threshold 100% of the time regardless of the number of particles used. In this case, the new topology on average led to fewer average iterations, though that is the only case that holds true for in the Rastrigin runs. For the CPSO-S_k tests, the Spatially Meaningful Neighbors topology consistently led to better results having one more successful run on each test, though requiring a higher average number of iterations to do it. The CPSO-R_k run shows the first time that the new Topology has led to a worse result, however on average, there is still a net benefit to the new topology due to the 16% increase in success percentage for the 15 particle run. These results are mirrored in Figure 5 as in most cases the Delaunay Triangulation version of the algorithm consistently has the lower fitness value of the coarse of each iteration.

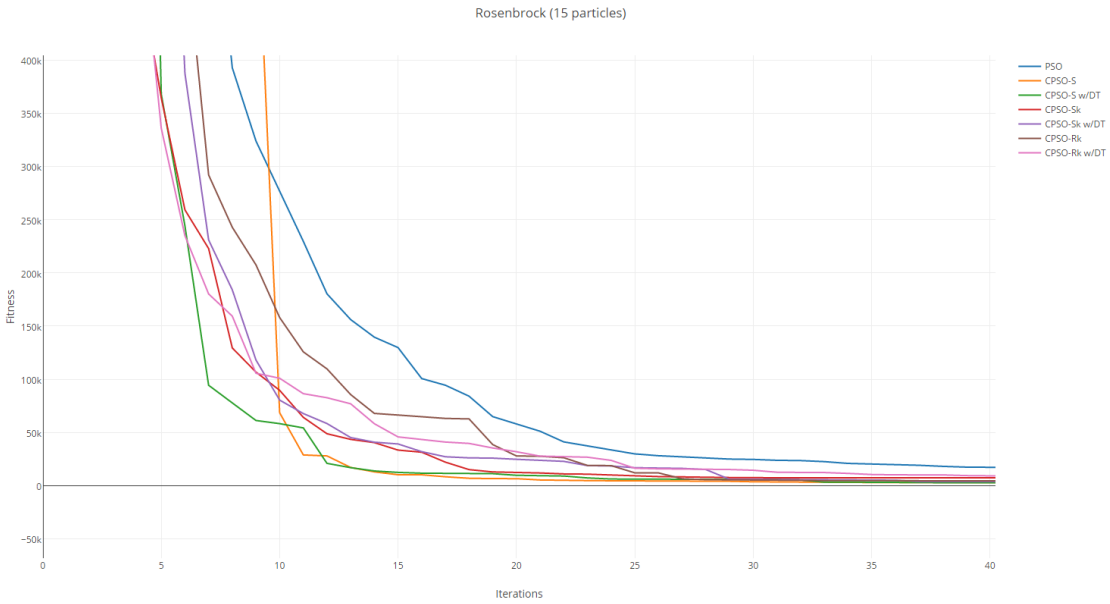

Figure 6: Rosenbrock Comparison (n = 6, particles = 15)

The results of the Rosenbrock tests are represented in Table 7. The table shows that, again, PSO has one of the higher average iteration counts but is much more successful than with the Rastrigin function. It was successful for almost all of their runs which is also true

Algorithm	s	Standard		Spatially Meaningful Neighbors	
		Succeeded	Iterations	Succeeded	Iterations
PSO	15	98%	346		
	20	100%	308		
	25	100%	350		
CPSO-S	15	100%	287	100%	356
	20	100%	128	100%	116
	25	100%	106	100%	96
CPSO-S$_3$	15	100%	284	100%	316
	20	100%	153	100%	156
	25	100%	90	100%	85
CPSO-R$_3$	15	94%	301	98%	386
	20	98%	392	100%	415
	25	100%	260	100%	275

Table 7: Rosenbrock Robustness Analysis (n=6)

for all of the other algorithms tested. For both the CPSO-S and CPSO-S$_k$ algorithms, the Spatially Meaningful Neighbors topology leads to a higher iteration count for the 15 particle test, however leads to similar or fewer iterations when using a greater number of particles. In this test, the CPSO-R algorithm performs the worst, however is improved upon with the new topology. With the use of Spatially Meaningful Neighbors, CPSO-R$_k$ is able to match the success rate of the PSO algorithm, however, in all cases it requires more iterations than the standard CPSO-R$_k$ and on average more iterations than PSO. Figure 6 shows this behaviour as again PSO is the slowest to converge while CPSO-R$_k$ is more often hanging on local minimums and requiring a number of iterations to break out.

The Griewank result in Table 8 show the largest boost from the Spatially Meaningful Neighbors topology yet. Once again, the PSO algorithm is the least successful, unable to get above 50% success rate while still requiring over 1200 iterations. The CPSO-S runs show both an increase and a decrease in success rate for different particle tests. On average, however, the success rate is the same for both topologies. The CPSO-S$_k$ and CPSO-R$_k$ algorithms see the largest benefit to the new topology here. Each test receives around a 20% boost in success rate at the cost of much higher iteration counts (an increase of around 200

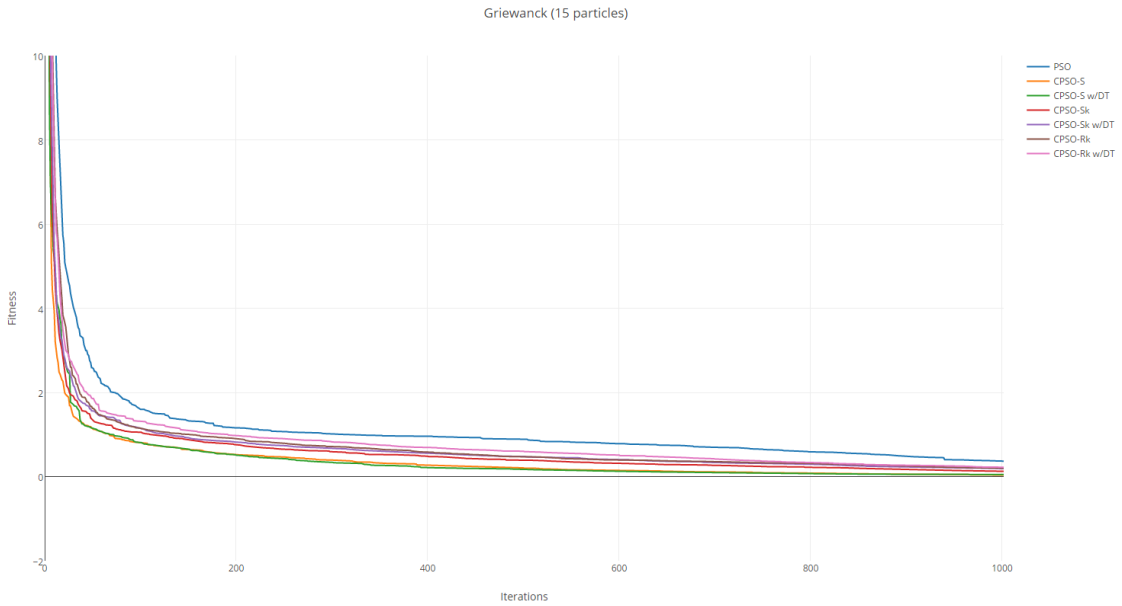

Figure 7: Griewanck Comparison (n = 6, particles = 15)

Algorithm	s	Standard		Spatially Meaningful Neighbors	
		Succeeded	Iterations	Succeeded	Iterations
PSO	15	40%	1323		
	20	28%	1269		
	25	22%	1222		
CPSO-S	15	98%	980	100%	976
	20	100%	724	96%	759
	25	98%	647	100%	733
CPSO-S_3	15	82%	1220	98%	1465
	20	84%	1208	100%	1440
	25	94%	1104	100%	1396
CPSO-R_3	15	62%	1301	82%	1531
	20	66%	1247	86%	1581
	25	80%	1230	98%	1700

Table 8: Griewanck Robustness Analysis (n=6)

each test). Looking at Figure 7, it shows that the Delaunay Triangulation version of the algorithms are consistently performing better than their standard counterparts throughout most of the run.

The Ackley test, however, sees the least change between topologies (as seen in Table 9). Throughout the table, the algorithms are almost identical. Both success rates and average

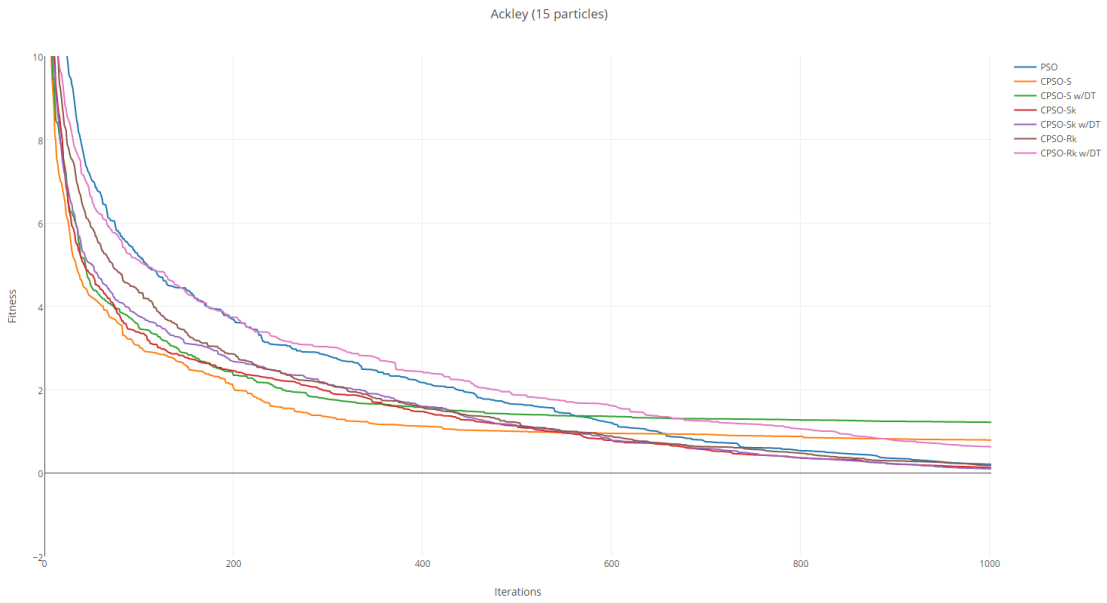

Figure 8: Ackley Comparison (n = 6, particles = 15)

Algorithm	**s**	**Standard**		**Spatially Meaningful Neighbors**	
		Succeeded	**Iterations**	**Succeeded**	**Iterations**
PSO	50	100%	1375		
	75	100%	1287		
	100	100%	1262		
CPSO-S	50	94%	1223	94%	1272
	75	100%	1115	100%	1137
	100	100%	1014	100%	1013
CPSO-S_3	50	100%	1261	100%	1263
	75	100%	1202	100%	1237
	100	100%	1148	100%	1189
CPSO-R_3	50	96%	1351	94%	1365
	75	100%	1309	100%	1299
	100	100%	1211	100%	1267

Table 9: Ackley Robustness Analysis (n=6)

iteration counts see little to no change with the exception of the CPSO-R_k algorithm which displays a small decrease in success rate when using the Spatially Meaningful Neighbors topology. In the graph at Figure 8 the 2 versions of the algorithms tend to mirror each other and stay close to their topology counterpart.

4.4.2 Discussion

The Robustness metric shows how successful the swarms are at finding optimal solutions and, in comparison, the effect the Spatially Meaningful Neighborhood Topology has on this success value. In most cases, we can see a positive effect of utilizing spatial information in determining the movement of the particles. The trade off to this is the spatially meaningful neighbor tests tend to take more iterations before they converge on the threshold. This is an expected side effect since it opts for more exploration. This trade off can range from a single extra iteration to hundreds. The benefits can be seen in a number of cases, especially for the Griewanck function (Table 8), there is a large boost of 10%-20% coming from the change in the neighborhood topologies. This, however is not always the case. There are a number of instances where the new topology leads to around 2% fewer successes (or one extra failure). These discrepancies can often potentially be explained by the random nature of the initial particle generation.

4.5 Experiment 2: Robustness Test on Higher Dimensions

The issue with the Delaunay Triangulation method for calculating Spatially Meaningful Neighbors, as originally pitched, was that the algorithm did not scale well to higher dimensions.[2] When the dimensions reached 4 or more, the process to calculate this Triangulation becomes significantly more complex and time consuming. This excess calculation negates the benefits that come from the new topology since it no longer becomes a viable option. By limiting the Delaunay Triangulation calculations to just 3-dimensions, the number of calculations required to create the triangulation drops to just $O(n^2)$, which is significantly more manageable than the dimension dependant exponential problem it otherwise would be. In the following test, the Delaunay Triangulation method is used in conjunction with CPSO to attempt to still gain the benefit of the topology while solving problems of dimension 20. The number of particles and swarms were uniformly increased with the dimension to ensure that all CPSO variants have enough particles to adequately search the search space. CPSO-

S, for instance, would then need to be divided into 20 swarms so ensuring each swarm had at the very least, 2 particles was key. If the method can scale to larger dimensions, it may be a viable solution for a greater number of problem.

4.5.1 Results

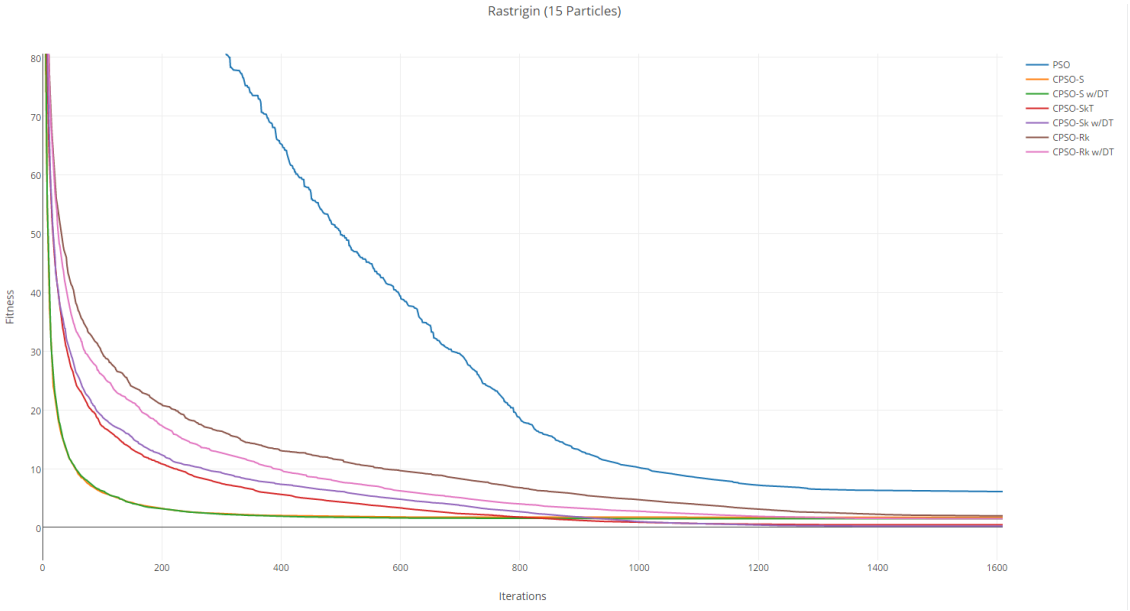

Figure 9: Rastrigin Comparison (n = 20, particles = 50)

Algorithm	s	Standard		Spatially Meaningful Neighbors	
		Succeeded	Iterations	Succeeded	Iterations
PSO	50	28%	1434		
	75	44%	1351		
	100	62%	1236		
CPSO-S	50	94%	986	94%	1012
	75	100%	800	100%	827
	100	100%	639	100%	624
CPSO-S_3	50	76%	1258	86%	1321
	75	86%	1158	90%	1287
	100	100%	1047	100%	1238
CPSO-R_3	50	26%	1458	34%	1324
	75	56%	1244	56%	1548
	100	68%	1485	74%	1152

Table 10: Rastrigin Robustness Analysis (n=20)

By increasing the number of dimensions, as seen in Table 10, PSO is shown to have a harder time converging onto the threshold without a large number of particles. For all but the 100 particle test the PSO performs worse than its lower dimension counterpart despite the same particles to dimensions ratio. Of the tested algorithms, the standard PSO performed the worst. The CPSO-S algorithm performed very similarly to the smaller dimension counterpart though it did not see any benefit from the use of the new topology. CPSO-S_k and CPSO-R_k, however, both saw consistent improvements in success rate when using the Spatially Meaningful Neighbors topology. This led to around a 10% increase on each test and was able to bring the CPSO-S_k algorithm performing around the same success rate as the lower dimensions. The CPSO-R_k algorithm also struggled in higher dimensions, however, and without the new topology had similar results to the standard PSO. In the accompanying graph (Figure 9) the struggle for the standard PSO is clear. It converges significantly slower than the other algorithms. Additionally, it shows the Delaunay Triangulated algorithms all surpassing their counterparts at around the 1000 iteration mark.

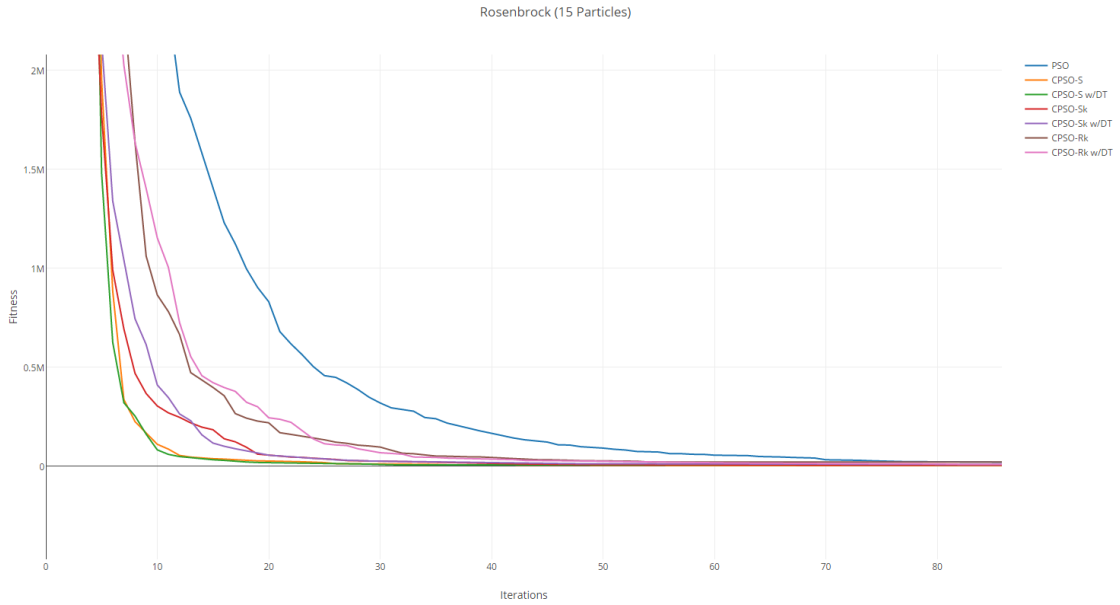

Figure 10: Rosenbrock Comparison (n = 20, particles = 50)

Table 11 details the results from the high-dimension Rosenbrock tests. In this case, the results are very similar to the lower dimension tests. All algorithms are able to consistently

Algorithm	s	Standard		Spatially Meaningful Neighbors	
		Succeeded	Iterations	Succeeded	Iterations
PSO	50	100%	366		
	75	100%	300		
	100	100%	250		
CPSO-S	50	100%	653	94%	605
	75	100%	382	100%	393
	100	100%	259	100%	187
CPSO-S_3	50	100%	622	96%	596
	75	98%	517	100%	545
	100	100%	465	100%	454
CPSO-R_3	50	100%	625	100%	704
	75	100%	542	100%	708
	100	100%	419	100%	611

Table 11: Rosenbrock Robustness Analysis (n=20)

hit 100% success rate for both topologies. In this case, the standard PSO algorithm actually performs the best, requiring on average the least number of iterations to achieve the desired success rate. In this case, the new topology had either no effect or in some cases a negative effect. On 2 separate occasions, the new topology lowered the success rate for the test runs, while also taking similar or more iterations to achieve that. The graph at Figure 10 shows the PSO is the slowest to converge but is ultimately able to pass the other algorithms to hit the threshold the quickest.

The table for the Griewanck results (Table 12) in the high dimension test shows the largest difference compared to the 6-dimension tests. In this case, the standard PSO went from the least successful to the most, being the only algorithm to achieve a 100% success rate. All other algorithms had a drop in success rating though the benefit gained from the new topology remained consistent here. All test either had similar or better results from using the Spatially Meaningful Neighbors topology though that still wasn't enough to match the results from the previous experiment and the graph (Figure 11) shows this. It demonstrates very few of the algorithms even getting close to the optimal solution, converging early and having trouble improving.

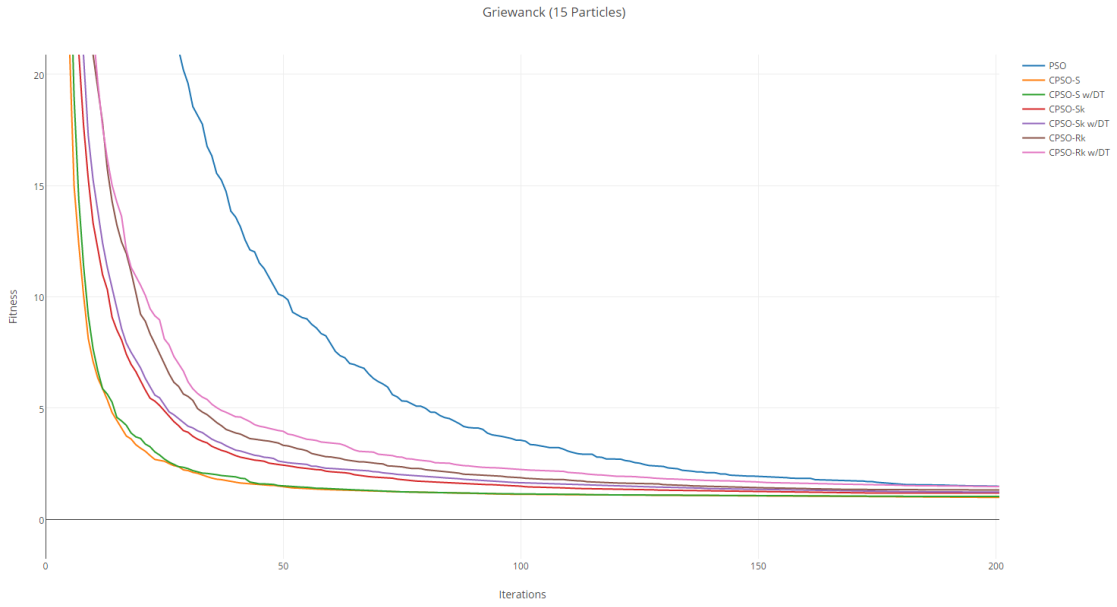

Figure 11: Griewanck Comparison (n = 20, particles = 50)

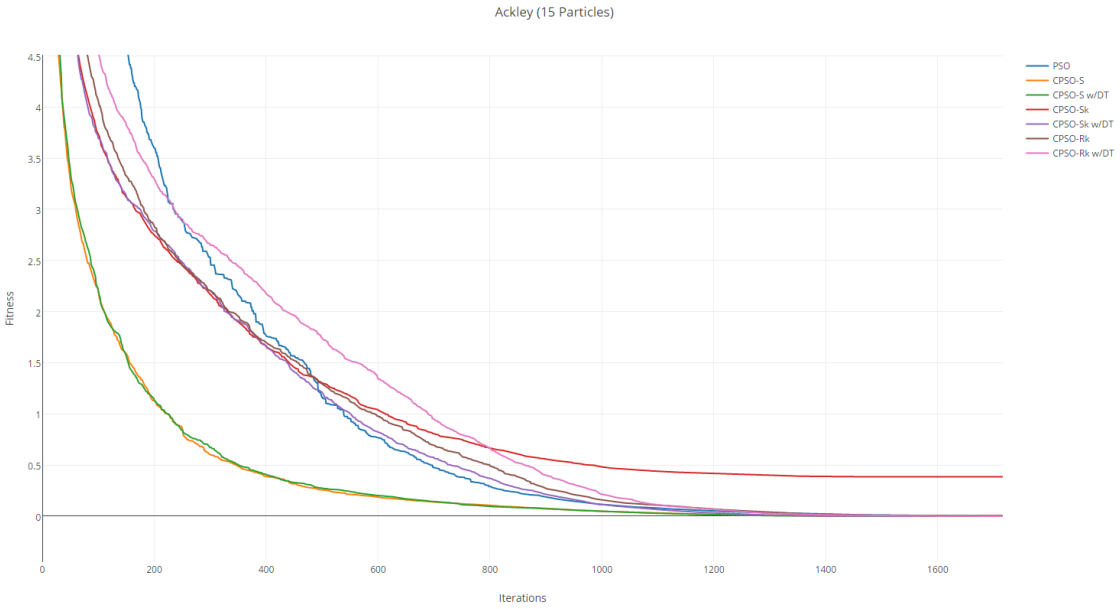

Figure 12: Ackley Comparison (n = 20, particles = 50)

Algorithm	s	Standard		Spatially Meaningful Neighbors	
		Succeeded	Iterations	Succeeded	Iterations
PSO	50	92%	1240		
	75	100%	1118		
	100	98%	1064		
CPSO-S	50	88%	1016	89 %	1052
	75	76%	939	76%	962
	100	74 %	705	80%	714
CPSO-S_3	50	78%	1256	78%	1415
	75	78%	1206	94%	1487
	100	84%	1136	94%	1486
CPSO-R_3	50	42%	1588	44%	1588
	75	50%	1286	52%	1570
	100	40%	1240	64%	1689

Table 12: Griewanck Robustness Analysis (n=20)

Algorithm	s	Standard		Spatially Meaningful Neighbors	
		Succeeded	Iterations	Succeeded	Iterations
PSO	50	96%	1507		
	75	100%	1387		
	100	100%	1295		
CPSO-S	50	100%	1254	100%	1262
	75	100%	1168	100%	1147
	100	100%	997	100%	992
CPSO-S_3	50	98%	1327	100%	1311
	75	100%	1236	100%	1226
	100	100%	1127	100%	1178
CPSO-R_3	50	94%	1426	100%	1426
	75	100%	1290	100%	1318
	100	100%	1217	100%	1246

Table 13: Ackley Robustness Analysis (n=20)

The final test was to optimize the Ackley function and the results are in Table 13. This test remains pretty consistent to the previous experiment. Similar to the lower dimension test, the majority of the runs we successful. This time, there was a more consistent increase for the Spatially Meaningful Neighbor topology as it was more successful in hitting the threshold. It was also able to do so in very similar iterations as its counterpart. Figure 12 shows just how successful the new topology was in this case. The CPSO-S_k algorithm without

Delaunay Triangulation performed significantly worse than version with that calculation. The other algorithms converge quickly to the threshold and are able to consistently be successful.

4.5.2 Discussion

After promising results in lower dimensions, tests were performed to see how well the algorithm scales to higher dimension problems. The problem set was increased from 6-dimensions to 20-dimensions to see the effect this has on the algorithms 'robustness'. The results show something of a decrease in the benefit from the Topology though still leading to on average better results. Part of the reason for this is the increase in particles, which led to all the variants having an adequate number to optimize the solution. The cases where it wasn't 100% saw a small increase in effectiveness but not a large enough boost the definitively conclude that it was solely due to the use of spatially meaningful neighbors. It must also be noted that the run-time of the Delaunay Triangulation version of the algorithms also ran exceedingly slow. This is mostly due to the number of times we are calling that Delaunay Triangulation (DT) algorithm. Since the swarm is divided in n/2 or n/3 sub swarms, each of which needs to calculate the DT with each iteration. This leads to an extra $O(n * n^2)$ algorithm being run every iteration of the CPSO. Though it is not as time consuming as $O(n^{\lceil d/2 \rceil + 1})$, especially on significantly large dimensions, it is also not an insignificant use of resources.

5 Conclusion

This paper examined the work brought fourth by Lane, Engelbrecht and Gain in search of improving the PSO algorithm by utilizing positional data to intelligently decide the movement of the particle, [2] and attempts to expand the work's reach by expanding it to new problems. In removing the initial dimensional limitation through the incorporation of coop-

erative swarms, this paper hopes this technique could be brought to larger more complicated problems.

The results of the study proved the benefits of this technique still hold up under the use of cooperative particle swarm optimization, though potentially at a diminished rate. By separating high dimension problem into multiple smaller problems, the Delaunay Triangulation calculations required for defining the distance between particles becomes significantly more manageable and actually feasible in appropriately large problems. However, although this change significantly decreases the overall number of calculations needed, there is still a large overhead required for this calculation. Since it needs to be calculated for every swarm on every iteration, this can dramatically slow down the run time of the application. Additionally, the benefit gained from this technique is highly dependent on the problem at hand. Where some of the tests resulted in significant success rate boosts, other problems experienced little to no gain, making the large overhead hard to justify. Ultimately, if the success of the algorithm is of higher priority than the time it takes to complete, the Spatially Meaningful Neighbor topology is certainly worth a look.

6 Future Work

This paper only touches on the potential of new neighborhood topologies that can come from this general idea. Having particles be more aware of their position in relation to the rest of the swarm is ripe for improvements. Testing out a number of different knowledge transfer techniques along with the dynamic nature of this neighborhood topology can lead to a more intelligent particle. Potentially having particles more directly communicate with the swarm as a whole would be invaluable.

Further work could also be invested in discovering a quicker, less time consuming way of determining the neighbors. The Delaunay Triangulation algorithm adds a large amount of overhead, especially for a calculation that is being repeated as often as it is. Even in lower

dimension problems, the Delaunay Triangulation algorithm is the bottleneck calculation and significantly slows down execution of the Particle Swarm Optimization.

Finally, work could also be invested in adding this technique to different variants of the CPSO algorithm. For the purposes of the preliminary tests of this paper a number of the CPSO variants were note tested here. A potential next step could be to try it to more and see if there is a difference in the effect it has on other variants, such as the mentioned CPSO-H_k algorithm.

References

[1] J. Blackwell R. Kennedy. Particle swarm optimization. *Swarm Itelligence*, 1:33–57.

[2] Andries Engelbrecht James Lane and James Gain. Particle swarm optimization with spatially meaningful neighbours. *Swarm Intelligence Symposium*, September 2008.

[3] Mu˜noz Zavala Angel Eduardo. A comparison study of pso neighborhoods. *EVOLVE - A Bridge between Probability, Set Oriented Numerics, and Evolutionary Computation II*, 175:251–265.

[4] Gjacquenot. Delaunay circumcircles vectorial. [Online; accessed May 3 2016].

[5] Jose Gabriel Ramires-Torres Angelina Jane Reyes Medina, Gregorio Toscano Pulido. A comparative study of neighborhood topologies for particle swarm optimizers.

[6] D. T. Lee and B. J. Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer and Information Sciences*, 9:219–242.

[7] F. Van den Bergh and A.P. Engelbrecht. A cooperative approach to particle swarm optimization. *IEEE Transactions on Evolutionary Computation*, page 225–239, June 2004.

[8] Alan Mackworth David Poole and Randy Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, 19986.

[9] Magnus Erik Hvass Pedersen. Good parameters for particle swarm optimization.

[10] Justin Maltese. Vector-evaluated particle swarm optimization using co-operative swarms. *3F90 Thesis*, 2014.

[11] Hanning Chen Tao Ku Wenping Zou, Yunlong Zhu. Clustering approach based on von neumann topology artificial bee colony algorithm.

[12] Lena Schlipf Panos Giannopoulos, Wolfgang Mulzer. Computational geometry: Polygon triangulation.

[13] M.A. Potter and K. A. de Jong. A cooperative coevolutionary approach to function optimization. *The Third Parallel Problem Solving From Nature*.