CPSO With Spatially Meaningful Neighbors

Peter Wilson
Department of Computer Science
Brock University
Email: peterwilson66@gmail.com

Prof Ombuki Berman
Department of Computer Science
Brock University
Email: bombuki@brocku.ca

Prof Andries P Engelbrecht
Department of Computer Science
University of Pretoria
Email: engel@driesie.cs.up.ac.za

***Abstract*—Particle Swarm Optimization (PSO) is a population-based computational intelligence algorithm to optimize continuous functions [1] through communication among its particles. This communication often works is via naive, static communication channels. One improvement would be to utilize the dynamic distances of particles to dictate communication with particles that are positionally close. However, such a technique would require intensive calculations to determine these distances and is therefore not viable for larger dimension problems. This proposes and tests an alternative version, showing that using cooperative swarms to divide up higher dimension problems makes this new form of communication possible for more complex problems.**

I. Introduction

Particle Swarm Optimization is a population-based computational intelligence algorithm based on the concept of birds flocking patterns [1]. The power of population allows it to find optimal solutions to a problem over multiple generations. By randomizing potential solutions (particles) and continuously changing them based on information already determined, the swarm is able to converge on solutions without an exhaustive search. As such, communication between particles is key. The standard form of communication (follow the leader) has the overall best solution telling the rest of the particles to follow. As the PSO algorithm matured, different communication methods have been introduced [2].

One method, called spatially meaningful neighbors, is to have particles communicate only if they are searching the same subspace [3] (pockets of search space that have their own local optimal value), allowing for more cooperation and a more thorough search of this space. To determine communication, distances between each particle would be calculated. This quickly becomes a bottleneck as dimensions increase past 3-dimensions since calculating the distance for each iteration can become incredibly time consuming [4].

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 Cooperative PSO [5] does exactly that, breaking dimensions down into smaller problems that collaborate to solve the main problem. By using CPSO in conjunction with the spatially meaningful neighbors, this thesis aims to bring that technique to higher dimension problems while still maintaining the subspace searching benefits it offers.

II. Background

A. Particle Swarm Optimization

1) Overview: Particle Swarm Optimization (PSO) is an iterative population-based computational intelligence algorithm which searches for optimal solutions to 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 [6]. PSO 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 converge toward that point.

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; updating 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,... coordinates of the position also correlate to the input values in the problem which we are trying to optimize. The greater number of input values (variables) the problem contains, the higher dimensions are needed to represent the problem. The initial position of the particle is randomly generated to allow the swarm to effectively test the entire search space. The velocity represents the direction and speed of the particle, dictating the new position after each iteration. The velocity is also updated at the end of each iteration, and is influenced by: the current velocity, the personal best value (the best value previously found by the particle) and the global/neighborhood best (the best value found by all the particles neighbors). The velocity and position are updated using the functions below.

3) Position Update Formula:

$$x_i = x_i + v_i$$

Updating the position is as simple as taking the current position (x_i) and adding the current velocity (v_i) 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.

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 its 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 add more randomness and aid in keeping the swarm from converging too quickly.

5) Swarm: The swarm is the combination of a number of particles working in unison to solve the same problem. The particles are connected as a topology which allows them to share information allowing them to work together to converge on an 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 algorithm to accomplish this is in Table I.

TABLE I
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
```

Every iteration, the swarm will update the personal best for each particle (represented by $\vec{y}$) and update the global best if a new best value is found [1]. The value is calculated through passing the solution set into the fitness function (see below). Each particle is then iterated through to update velocity and position. It is important to complete all global/personal best updates prior to updating the velocities since those values influence the new velocity and therefore should be correct for those calculations.

6) Fitness Function: Finding an optimal solution to a problem first requires a way to evaluate individual solutions in relation to the optimization problem. The fitness function takes the list of input values which make up the solution and determines how well it fits the problem. This will return a single value that helps the swarm determine the success of the 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 PSO algorithm to solve different problems.

7) Parameters: Parameters can be altered to affect the success of the algorithm [1]. Changing 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 [7].

Max Iterations: How many iterations the swarm will go through in to search for the optimal solution. Greater Iterations provides the algorithm with more time to converge to an optimal solution, though better algorithms should require less iterations to hit this target.

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

Inertial Weight (α): Determines the influence the current velocity has on future velocities. A large inertial weight will stubbornly lead to the particle on its current direction allowing for greater exploration. A low inertial weight will give more influence to the communication, allowing for greater exploitation.

Cognitive Weight (ω): 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): Determines the effect neighborhood particles (global best) have on the current particle. Different neighborhood layouts/topologies can change the effect of the social communication (See II-C). A low value will have the particles mostly work on their own, whereas a large social weight will increase exploitation.

B. Co-operative Particle Swarm Optimization

For a problem with a large enough number of inputs, PSO can struggle as it can be difficult for the swarm to hone in on an optimal solution given the breadth of the search space size, often getting stuck in local optima. This is known as the curse of dimensionality [5], which is a side effect of all the dimensions updating at once. This can result in a situation where a new solution is worse for most dimensions but leads to a better fitness value overall due to a better value for a single dimension [8]. For instance, for minimizing 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 [6,1,7] returns a better value of 14 though the first and last dimension values are moving away from the optimal solution [8]. This can influence the velocity, leading to the particles heading in the wrong direction.

CPSO attempts to solve the curse of dimensionality by partitioning the problem into multiple sub-problems [5], allowing each subset of the dimensions to be solved separately with their own swarm. Each iteration will test its solution along with static random values in place of the other inputs. Different variants of the algorithm introduce alternative ways of partitioning the larger problem. An issue which arises from this partitioning where it can remove potential input dependencies from the original problem. If two dimensions

are required to work together to improve the fitness, they will be no longer able to do so directly if partitioned to different swarms.

1) CPSO-S: The original version of CPSO is called CPSO-S. Developed by Van den bergh and Engelbrecht [5], it tackles the problem by dividing an n-dimensional problem into n 1-dimensional swarms. The algorithm will iterate through each single dimension swarm, operating a standard PSO for each partition (as seen in Table II).

TABLE II
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.(x)
            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 PSO, separating the problem into 1-dimensional problems significantly simplifies the end goal. However, since each swarm operates almost in a vacuum, it removes all variable dependencies from the original problem, which can make it difficult to find the optimum solution.

2) Variant: CPSO-S_k: The first variant proposed by Van den bergh and Engelbrecht is CPSO-S_k[5]. It evenly divides the problem into k swarms (or as close to even as possible). This attempts to combine dependent variables to keep them intact. The rest of the algorithm is identical to CPSO-S (See Table II). Though this does give the potential to combine dependent terms, it still does so blindly, therefore there is no guarantee.

3) Variant: CPSO-R_k: Another potential solution to the partitioning problem would be to randomly partition into a set number of blocks, which is implemented in CPSO-R_k (also known as Random CPSO). This can lead to a larger likelihood of combining dependant variables depending on block size (k). If it does successfully make those connections it helps the chances for a successful run. Once the partitions are established, the CPSO-S algorithm is run to solve the PSOs (See Table II).

C. Neighborhood Topologies

The neighborhood topology of a PSO dictates how particles are allowed to communicate with each other, allowing them to work together in solving the problem. This communication is achieved through the global best values within the velocity update function. In the standard PSO (Section II-A5), all particles are able to communicate with each other, so all particles in the swarm are led by the overall best particle in the entire swarm. This behaviour can lead to naive behaviour that converges on an optimal solution without fully exploring the search space. To combat this, a local PSO [9] was proposed, which utilizes different, more constrained, neighborhood topologies. A visual representation of these topologies can be found on Figure 1.

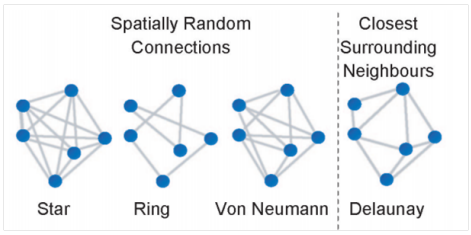

Fig. 1. Visual Representation of the Neighborhoods[3]

1) Common Spatially Random Topologies: A variety of neighborhood topologies are currently being used and tested in the PSO community. These topologies have their own pros and cons and can lead to different results depending on the problem in question. Spatially random topologies are ones which do not take particle location into account to create the topology. [3].

Star

The star topology is used in the standard PSO. It connects every particle with each other, meaning the neighborhood best for each particle is the global best position in the swarm. Star topology often converges quickly onto a value without a thorough exploration of the search space.

Ring

The ring topology changes the topology by reducing the number of connections for particle to just two each in a chain-like pattern. This slows the transfer of information, decreasing the global bests impact on the swarm[9] and helping slow its convergence. This should allow for a more thorough investigation of the search space, taking more iterations to do so. Connections are often determined randomly during the initialization of the particles, though, which means 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, increasing convergence rate in the process, while still maintaining the slow information transfer rate within the network[10]. Particles are connected in a cubic-lattice arrangement [3] where particles are connected on the left, right, above and below. This is not determined by coordinate location, but often based on index value in the swarm array. Above and below are based on an offset determined by swarm size, meaning it does not take advantage of positioning information and connections are not updated throughout the

completion of the run.

2) Spatially Meaningful Neighbors : The success of spatially random topologies are ultimately dependent on the success of the initial randomization. Since the topology essentially guides the particles, if connections were dynamically chosen based on found information, a stronger collaboration between particles could be formed. Lane, Engelbrecht and Gain proposed [3] using coordinate distance between particles to determine these connections. It was theorized that particles close together have a higher probability to be searching the same subspace, allowing them to work together to examine the space. The proposed new topology, called spatially meaningful neighbors, uses Delaunay triangulation to connect all particles with the neighbors that are closest (See Section II-D). The tests found this leads to a more guided search and a higher success rate [3].

D. Dynamic Connections

In addition to dynamic topologies, the use of dynamic connections could also be of help. By determining which of the neighboring particles are working together, connections could be made to improve cooperation for searching subspaces [3]. This is accomplished by replacing the neighborhood best value in the velocity update with the personal best of the particle it is working with, forcing those two particles to influence each other. An example of particles working together can be found in Figure 2. The criteria for a particle to be working with another are:

- Particle P_1 is following behind particle P_2. i.e. both are heading in the same direction. This this case, a directed connection will be established so the particle in front can send information to the trailing particle[3]
- 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 [3].

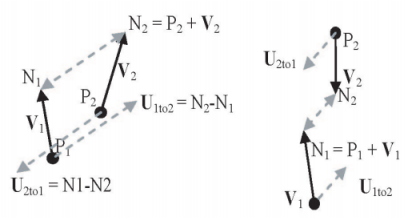

Fig. 2. How to know if the particles are working together[3]

Both cases can be tested 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 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.

Delaunay triangulation, however, 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 III details the process of finding the neighbor to communicate with. In this algorithm P_f represents the personal best position of the fittest connected neighbor. P_c represents the personal best position of the closest fitter connected neighbor. The idea behind the code below is that if the overall fittest neighbor is too far to be considered as searching the same subspace (in this case, is further than our defined $localExploitationRatio$), then the algorithm will instead choose the closer of the neighbors as long as the fitness value is better than its own.

TABLE III
CHOOSING THE BEST NEIGHBOR

input: N_i particle is closest neighbors
procedure CHOOSEBESTNEIGHBOR(N_i)
  hasConnectedNeighbors = false
  $P_f \leftarrow min(N_k)$
  for $k \leftarrow 1 to neighborsetsize$ **do**
    if $workingtogether(P_i, P_k)$ 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
  $localExploitationRatio \leftarrow swarm.diameter/200$
  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

E. Delaunay triangulation

1) Triangulation: Triangulation is the decomposition of a set of points into a set of non-intersecting triangles[3] (or simplices in n-dimensions) [11]. 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. Optimal triangulation minimizes the total length of all edges, otherwise known has having the minimum weight. This means that an optimal triangulation will connect all the points to their nearest neighbors. Finding the optimal triangulation of a set of points is an NP-Hard problem [6], however, there are a number of triangulation algorithms that attempt to approximate optimal.

2) Delaunay triangulation: Delaunay triangulation (DT) is a triangulation approach that avoids sliver triangles (thin triangles not often found in minimum weight triangulations)

through maximizing the minimum angle of all the triangles[6]. This triangulation methodology creates a set of triangles such that the circumcircle of each triangle contains no other points (See Figure 3). This means 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 [3].

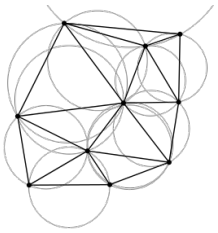

Fig. 3. Circumcircles drawn amongst the Triangles[12]

The issue with Delaunay triangulation is one of dimensionality. In 2-dimensions, Delaunay triangulation can be done with a time complexity of just O($nlogn$), one of the quickest methods for pseudo-optimal triangulation [3]. However, it becomes increasingly difficult to compute as dimensions increase, making it a less viable solution. In 4-dimensions the best case algorithm will have a worst-case O(n^3) run time. In d-dimensions, it takes O($n^{\lceil d/2 \rceil + 1}$) time.

III. 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. Such issues present in PSO were solved by utilizing CPSOs ability to break the problem into a number of smaller sub problems [8], therefore the same technique would be worth trying in conjunction with Delaunay triangulation. To take advantage of these benefits, all CPSO variations tested have been initialized such that no one swarm is handling more than 3 dimensions.

A. Experimental Setup

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

- **PSO**: For comparisons 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. In problems of a single dimension, the distances are calculated directly instead of using the Delaunay triangulation algorithm, due to the algorithm not working for 1 dimensions.
- **CPSO-S**$_k$: A k was 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.
- **CPSO-R**$_k$: The algorithm was altered to ensure all the random swarms are selected to only be between 1 and 3 dimensions. 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.

B. CPSO configuration

The number of iterations for CPSO and PSO implementations were capped at 10,000[5]. To ensure a fair comparison, the particles allotted to each algorithm were be distributed evenly amongst each swarm. This means that,for each iteration, the swarm will perform the same number of fitness calls to allow a 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 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**: evenly distributed among the swarms in the algorithm. i.e., if the particle count is set to 100 and the CPSO has 10 different swarms, each swarm will only have 10 particles to work with.

C. Benchmark functions

The functions tested against are below.

Function	Domain	Criterion	Min Coordinates
Rastrigin	[-5.12;5.12]	0.01	(0, 0, ...)
Rosenbrock	[-30;30]	100	(1, 1, ...)
Griewank	[-600;600]	0.05	(0, 0, ...)
Ackley	[-32;32]	0.01	(0, 0, ...)

TABLE IV
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. Below are the benchmark functions that are being tested as well as a description of the attributes of the function which makes each worth testing in this experiment.

1) Rastrigin:

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

The Rastrigin function has a large number of local optima throughout the search space intended to trap optimization

functions in incorrectly converging in a pseudo-optimal value. The further away from the origin, the larger the fitness values from both the peaks and valleys of the subspace pockets. Rastrigin is a good benchmark function as it helps determine how well the optimization algorithm is able to avoid getting stuck at pseudo-optimal solutions.

2) Rosenbrock:

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

The Rosenbrock function takes on the shape of a valley. Though locating the valley within the function is trivial, the difficulty is in finding the optimal value within it. The global minimum is inside a long, narrow, parabolic shaped valley and resides at the coordinate when all inputs are 1.

3) Griewank:

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

The Griewank function follows a similar pattern as the Rastrigin function above. It contains pockets of local optima, however,where the Rastrigin function features steep/narrow peaks, the Griewank contains subspaces that are larger and require the algorithm to explore more to escape the local optima.

4) Ackley:

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

The Ackley function has a similar set of local optimal subspaces, however, the pockets remain consistent, making it more difficult to surmise which direction to head. Ackley has a large hole in the center of the graph that leads straight to the global best value. If the optimization function is able to find this hole, it should have no problem converging on the optimal solution.

D. Experiment 1: Robustness Test on Lower Dimensions

A good PSO algorithm is one that can quickly reach the optimal result on a consistent basis. It is then important that the robustness of the algorithm be tested to successful it is. In this case, robustness refers to the percentage of successful runs and the iterations required to reach success. A successful run is one that reaches the solution threshold before the max iterations. The threshold varies per function (see III-C) but the max iterations is 10,000 iterations for all. To make the iterations equivalent for all tests, the number of particles per algorithm is divided evenly amongst swarms. This ensures each iteration evaluates the same number of particles regardless of swarm count. The result table has the columns in order: algorithms tested, particle count, star topology (with percent succeeded and iterations to success), and spatially meaningful neighbors in the same format. Note: if a run was not successful, the iteration count does not factor into the average iteration count. For this first experiment, tests were run against problems of 6-dimensions to see if the Delaunay triangulation calculations led to noticeable improvements.

Algorithm	s	Standard		Meaningful Neighbors	
		Succeeded	Iteration	Succeeded	Iteration
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 V
RASTRIGIN ROBUSTNESS ANALYSIS (N=6)

1) Results: Table V details the results of the Rastrigin function. Of the algorithms tested, the standard PSO performed the worst in both metrics. CPSO-S converged on the threshold 100% of the time regardless of particle count. The new topology on average led to fewer iterations. In CPSO-S$_k$ tests, the spatially meaningful neighbors topology had one more successful run on each test, requiring more iterations to do it. CPSO-R$_k$ has a case where the new topology 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.

Algorithm	s	Standard		Meaningful Neighbors	
		Succeeded	Iteration	Succeeded	Iteration
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 VI
ROSENBROCK ROBUSTNESS ANALYSIS (N=6)

the Rosenbrock tests (represented in Table VI) show PSO is again the worst performing,taking longest to converge. Both the CPSO-S and CPSO-S$_k$, see similar results with each topology. Here, CPSO-R performs the worst with the star topology but sees an improvement with the new topology, even if it still requires more iterations than the others.

The Griewank result in Table VII shows a largest boost from the new topology. Again, PSO algorithm is the least successful while CPSO-S shows mixed results between topologies. The CPSO-S$_k$ and CPSO-R$_k$ algorithms, however, see the largest benefit with the new topology. Each test receives around a 20% boost in success rate at the cost of more iterations.

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

Algorithm	s	Standard		Meaningful Neighbors	
		Succeeded	Iteration	Succeeded	Iteration
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 VII
GRIEWANK ROBUSTNESS ANALYSIS (N=6)

Algorithm	s	Standard		Meaningful Neighbors	
		Succeeded	Iteration	Succeeded	Iteration
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 VIII
ACKLEY ROBUSTNESS ANALYSIS (N=6)

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.

2) Discussion: Robustness shows how successful the swarms are at finding optimal solutions and, in comparison, the effect the spatially meaningful neighborhood Topology has on this success. In most cases, there is a positive effect of utilizing spatial information in the topology. The trade off is a greater number of iterations before convergence on the threshold. This is an expected side effect since it opts for more exploration. The benefits can be seen especially for the Griewank (Table VII) and Rastrigin functions (Table V), with a large boost of 10%-20% success rate. This shows the new topology is better at exploring sub-spaces since the particles are able to work together to investigate these spaces. There are a number of instances where the new topology leads to around 2% fewer successes (or one extra failure). These discrepancies can potentially be explained by the random nature of particle generation.

E. Experiment 2: Robustness Test on Higher Dimensions

The issue with Delaunay triangulation is it does not scale well to higher dimensions [3]. At 4 dimensions or higher, the calculation becomes exponentially more complex and time consuming. This negates the benefits of the new topology since it no longer becomes a viable option. Through limiting the Delaunay triangulation calculations to just 3-dimensions, the calculations required drops to a more manageable O(n^2). In the following test, the Delaunay triangulation method is used in conjunction with CPSO to gain the benefit of the topology while solving problems of dimension 20. The particle and swarm counts are uniformly increased to create comparable results to Experiment 1.

Algorithm	s	Standard		Meaningful Neighbors	
		Succeeded	Iteration	Succeeded	Iteration
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 IX
RASTRIGIN ROBUSTNESS ANALYSIS (N=20)

1) Results: After the dimension increase (Table IX), PSO struggles converging on the threshold, performing worse than the 6 dimension counterpart. CPSO-S performs similarly to Experiment 1, though not seeing a change between topologies. CPSO-S_k and CPSO-R_k, both experienced improvements with the new topology, leading to around a 10% increase, helping it match its success in Experiment 1.

Algorithm	s	Standard		Meaningful Neighbors	
		Succeeded	Iteration	Succeeded	Iteration
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 X
ROSENBROCK ROBUSTNESS ANALYSIS (N=20)

Table X details the Rosenbrock tests. The results are very similar to Experiment 1 as all algorithms consistently hit 100%. PSO performs the best here, requiring the least number of iterations. The new topology had little to no effect with 2 occasions of it actually lowering the success of runs.

The Griewank results (Table XI) show the largest difference compared to Experiment 1. PSO went from the least successful to the most, being the only algorithm to achieve a 100% success. All other algorithms showed a drop in overall success but still gained benefit from the new topology. All test either had similar or better results from using the new topology, showing significant gains.

Algorithm	s	Standard		Meaningful Neighbors	
		Succeeded	Iteration	Succeeded	Iteration
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 XI
GRIEWANK ROBUSTNESS ANALYSIS (N=20)

Algorithm	s	Standard		Meaningful Neighbors	
		Succeeded	Iteration	Succeeded	Iteration
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 XII
ACKLEY ROBUSTNESS ANALYSIS (N=20)

The Ackley function results in Table XII show the majority of the runs were successful. It details benefits from the new topology, being more reach 100% success while not using a much greater number of iterations.

2) Discussion: After promising results in lower dimensions, tests were performed to see how well the algorithm scales to higher dimension problems. The increase from 6-dimensions to 20-dimensions show a slight decline in benefits from the new topology though still leading to better results. A reason for this could be an insufficient particle count required to optimize. Cases where there was not 100% success experienced a small increase in effectiveness. CPSO algorithms performed the same or better than PSO in all but the Griewank algorithm. This is expected since it is known the function becomes easier to solve in higher dimensions [13], meaning the dimension split had a negative effect.

IV. CONCLUSION

This paper examined the work brought fourth by Lane, et al in improving PSO by using intelligent dynamic neighborhood topologies, [3] and extends it by expanding it to new problems. In removing the initial dimensional limitation via cooperative swarms, this paper tests if the technique could be brought to more complicated problems. The results proved the benefits still hold up with CPSO, as particles were able to work together to exploit sub-spaces. However, though this change significantly decreases the overall number of calculations needed, there is still a large overhead required. The high number of calculations each iteration can dramatically slow the run time of the application. Additionally, the benefit gained from this technique is dependent on the problem at hand. Functions that contained difficult sub-spaces experienced large boosts in performance. 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.

V. 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. Further work could also be invested in discovering a less computationally expensive approach for determining distance. Finally, work could also be invested in adding this technique to different variants of CPSO, such as CPSO-H_k (the hybrid alternative).

REFERENCES

[1] R. Poli, J. Kennedy, and T. Blackwell, "Particle swarm optimization," *Swarm Intelligence*, vol. 1, no. 1, pp. 33–57, 2007. [Online]. Available: http://dx.doi.org/10.1007/s11721-007-0002-0

[2] A. J. R. Medina, G. T. Pulido, and J. G. Ramrez-Torres, "A comparative study of neighborhood topologies for particle swarm optimizers." in *IJCCI*, A. D. Correia, A. C. Rosa, and K. Madani, Eds. INSTICC Press, 2009, pp. 152–159. [Online]. Available: http://dblp.uni-trier.de/db/conf/ijcci/ijcci2009.htmlMedinaPR09

[3] J. Lane, A. Engelbrecht, and J. Gain, "Particle swarm optimization with spatially meaningful neighbours," in *in Proceedings of the 2008 IEEE Swarm Intelligence Symposium. Piscataway, NJ: IEEE.* Press, 2008, pp. 1–8.

[4] D. T. Lee and B. J. Schachter, "Two algorithms for constructing a Delaunay triangulation," *International Journal of Computer & Information Sciences*, vol. 9, no. 3, pp. 219–242, 1980. [Online]. Available: http://dx.doi.org/10.1007/BF00977785

[5] F. V. den Bergh and A. Engelbrecht, "A cooperative approach to particle swarm optimization," *IEEE Transactions on Evolutionary Computation*, p. 225239, June 2004.

[6] D. Poole, A. Mackworth, and R. Goebel, *Computational Intelligence: A Logical Approach*. Oxford, UK: Oxford University Press, 1997.

[7] M. E. H. Pedersen, "Good parameters for particle swarm optimization," *Hvass Lab., Copenhagen, Denmark, Tech. Rep. HL1001*, 2010.

[8] J. Maltese, B. Ombuki-Berman, and A. Engelbrecht, "Co-operative vector-evaluated particle swarm optimization for multi-objective optimization," in *Computational Intelligence, 2015 IEEE Symposium Series on*. IEEE, 2015, pp. 1294–1301.

[9] A. E. Muñoz Zavala, *A Comparison Study of PSO Neighborhoods*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 251–265. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-31519-0_16

[10] W. Zou, Y. Zhu, H. Chen, and T. Ku, "Clustering approach based on Von Neumann topology artificial bee colony algorithm," in *2011 international conference on data mining (DMIN11)*, 2011.

[11] M. d. Berg, O. Cheong, M. v. Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Santa Clara, CA, USA: Springer-Verlag TELOS, 2008.

[12] Gjacquenot, "Delaunay circumcircles vectorial," https://upload.wikimedia.org/wikipedia/commons/d/db/Delaunay_circumcircle_vectorial.svg [Online; accessed 03-May-2016].

[13] M. Locatelli, "A note on the griewank test function," *Journal of Global Optimization*, vol. 25, no. 2, pp. 169–174, 2003. [Online]. Available: http://dx.doi.org/10.1023/A:1021956306041