Geographical Neighbourhood in Particle Swarm Optimization

Nicolas Höning

School of Computer Science & Informatics
University College Dublin

Abstract

In Particle Swarm Optimization (PSO) with local neighbourhood, the social part of change in the particle's velocity is computed considering the performance of a set of neighbours. Almost all of the literature uses neighbourhood relations of a fixed topology. This paper introduces a method that computes a local optimum based on geographical, non- fixed neighbourhood in Euclidian space. It compares the two approaches in performance and geographical behaviour. The results show that swarms with geographical neighbourhood perform worse in terms of fitness. Furthermore, the results indicate that swarms with fixed topologies start by exploring the search space due to initial random distribution and then turn to exploitation because of emerged geographical neighbourhood.

1. Introduction

Particle Swarm Optimization (PSO) is an algorithm with a natural inspiration. By guiding the movements of individual particles in a multi- dimensional space with simple rules, it was initially intended to simulate the behaviour of flocking birds. Only some time later it became clear that the multi- dimensional space could be viewed as a search space with each particle being a possible solution to the problem in question.
PSO resides in the same family of algorithms as Genetic Algorithms (GA), because both operate against a fitness function that describes the problem and update their population stepwise.
The update process of the particles velocities in the swarm constitutes the algorithm. Therein, each particle communicates the best solution found so far to neighbouring particles. This information, the so- called local best, is then incorporated into the update. The communication depends on the topology implemented in the swarm. If every particle is connected to every other particle, the local best is a global best. The research has been concentrated on topologies with fixed relations. That way, each particle will speak with the same neighbours over the whole course of the search, no matter where they are. I will implement a non- fixed neighbourhood that is defined by proximity in Euclidean space and compare the behaviour to a traditional, fixed topology.

The first hypothesis: With fixed neighbourhood, I expect the differences in neighbourhood distances to become smaller while the number of runs increases - a kind of geographical neighbourhood emerges in the traditional, fixed topology.

The second hypothesis concerns the first iterations of the algorithm: Comparing the two approaches, I suspect that with geographical neighbourhood, the exploration/exploitation – trade- off will tend to exploitation, because geographic neighbours explore the neighbourhood together (see section 2 for an introduction to the exploration/exploitation dilemma).
I figure that fixed neighbourhood is another factor of randomness that acts in favour of exploration: Since particles that are fixed neighbours tend to start on different locations in the search space (because they are randomly scattered there) they will - in

the first iterations- explore much of the room between them quickly, because the ones with bad solutions drift towards their best neighbour.

This paper is organized as follows: Section 2 introduces the PSO algorithm formally. Section 3 motivates geographical consideration of particle swarms and describes the algorithm I used to compute it. In section 4 and 5 I describe the experiment setup and results, respectively. Section 6 concludes the paper and provides an outlook on further work.

2. The PSO algorithm

Each particle in the swarm has a value for each dimension of the solution space and thereby describes its current position. Each particle also has a velocity for each dimension, describing its movement through the solution space. All those values are initialized randomly.
The algorithm is shown in formula (1). Its task consists in updating the velocity as the particles move through the solution space (With the new velocity *new_vel*, the new position *new_pos* can then easily be computed by adding the velocity vector *new_vel* to the vector of the current position, *cur_pos*).
The algorithm works by orienting each particle, for each dimension *d*, according to three things: its current velocity *cur_vel*, the best position it ever visited (*pbest)* and the best position one of its neighbours ever visited (*lbest*). Kennedy [5] calls orientation after *pbest* the 'cognitive' part, while orientation after *lbest* is called the 'social' part.

$$new\_vel_d = m{*}cur\_vel_d + (cc{*}cr{*}(cur\_pos_d - pbest_d)) + (sc{*}sr{*}(cur\_pos_d - lbest_d))$$

Formula (1) The PSO Algorithm

The algorithm can be tweaked with in many ways. The 'cognitive' and 'social' parts can be weighted via the constants *cc* and *sc*. They also get randomized by *cr* and *sr*.
To control the range of values in the particles, it's also common to specify an initialization range from which start- values are randomly chosen. Furthermore (not shown in formula (1)), a technique called “velocity clamping” can be applied: The values for the velocity are restricted to be in a given range. This protects particles from becoming too fast.

The PSO algorithm faces, like every search algorithm, the dilemma between exploration and exploitation. The particle swarm should explore much of the search space to avoid local optima. But it should also exploit promising solutions to find real good solutions. One way to address this problem is to limit the velocity of the particles by a momentum weight. In (1), this factor is represented by *m*. Some researchers prefer to set it to a value below 1, like 0.7. Others gradually decrease its value over the time of the run.
Another way of looking at the problem is the social part of the algorithm. The behaviour of a particle swarm includes an independent search by each particle as well as a communication within the swarm about good solutions. This communication also plays a role in the exploration/exploitation- dilemma. For instance, the faster the news of a good solution spreads within the population, the higher is the risk of it becoming a local optimum because other particles, attracted by the knowledge of the good solution, stop exploring for better solutions. Many different fixed topologies have been discussed (Kennedy, Mendes [6]; Richards, Ventura [7], Suganthan [8]). For example, the so- called “star”- topology, in which every particle knows about every other particles personal best, is mostly considered as able to reach good solutions quickly, but it is also in danger of converging on local optima.

The most-widely used fixed topology is the “ring”-topology (every particle with an index *i* has two neighbours, namely those with index *i*-1 and *i*+1). I will use it to compare fixed with geographical neighbourhood.

3. Geographical Neighbourhood

3.1 Motivation

While the original inspiration to the Particle Swarm Algorithm stems from nature, it's now mostly being used for optimization purposes. For that purpose, several methods can be applied to achieve efficiency, though they sometimes lead away from the original idea. In a swarm of insects (or birds, or fish) each particle will try to stay near to its geographical neighbours. In which way this exactly happens, is unclear. Still, the common method in PSO research is to define neighbourhood relations that are fixed for the whole search. Geographical relations have not yet been studied very much. The fixed neighbourhood is certainly an artificial situation. It's therefore interesting to study geographical neighbourhood to learn more about nature itself.
For example, geographical neighbourhood can be of use for behavioural models, to the extent of studying human behaviour in social science contexts (Kennedy [5] addresses it shortly). Psychological findings tell us that humans are influenced by peer groups. But peer groups are not fixed. Following this thought even further, if the values that the individual particles carry would resemble real properties, why would two particles listen to each other, even if they are not at all alike (recall that the properties of the particles are randomized at the beginning of the algorithm)? In nature, a great part of attraction between individual organisms can be explained by their similarities. (There has also recently been shown that humans are attracted towards mates with different traits, such as a different immune system. Nevertheless, mating is mostly about similar traits.)

Furthermore, if differences between the two approaches can be described and explained, we might learn more about fixed neighbourhood, the method widely used. For instance, if any of the hypotheses can be shown to hold, we can make interesting statements about the behaviour of PSO algorithms with fixed neighbourhood topologies that haven't been clear before. We might also learn more about fixed neighbourhood while we focus the discussion on geographical properties. In addition to the classical fitness measures, I will introduce measures that deal with geographic properties of the particles neighbourhood relations.

.

3.2 The Algorithm

Geographical distance in a multidimensional space can easily be computed by the Euclidean distance metric. Let *a* and *b* be two particles and *dims* the number of dimensions in the solution space. The Euclidean distance between *a* and *b* is defined by the square root of the sum of the squared distances of the particle's values per dimension (see Formula (2)).

$$euclid(a,b) = \sqrt{\sum_{d=1}^{dims}(a_d - b_d)^2}$$

Formula (2) The Euclidean distance metric

While the Euclidean distance measure is easy to understand, the algorithm to determine which of all the other particles possibly are neighbours has uncomfortable runtime properties.

The algorithm has to define, for a given particle, which particles have the smallest Euclidean distance to it. The simplest method would be to simply compute the distance to every other particle and sort the results with respect to Euclidean distance in ascending order. This method is exact, but resource-consuming. It compares every particle to every other particle, with a calculation that involves adding up every dimension. If we assume that the number of particles is much higher than the number of dimensions (this would be a reasonable setup), the problem complexity class is still $O(n^2)$. Also, recall that this calculation is only per iteration.
There are a number of heuristics one could apply to shorten the runtime of the algorithm. One idea might be to setup a radius in which to look for particles and constantly widen it, until enough possible neighbours are in. We could exclude particles even faster if we compare the distances dimension-wise, instead of computing the Euclidean distance (in two-dimensional space that could be visualized as a squared search window rather than a round search spotlight). A good radius to start with could be the distance to the nearest neighbour of the last iteration. The problem with this approach is that our data structure doesn't allow testing a region for particles. Particles have no vision. Unless we introduce a spatial-indexed data structure, we would compare all the particles again, ending up in the same complexity class.
We could also avoid comparing all particles by sorting neighbours in the array of particles next to each other once we identified them as neighbours. This would shorten the time to find near particles. But then we would lose exactness. We would only need to test some particles instead of all, but we might overlook others that are much nearer.
The implications of these heuristics for the behaviour of the PSO algorithm are unclear. It might be possible to be much more efficient and still achieve reasonable search behaviour. Since the goal of this paper is a comparison between geographical and fixed neighbourhood, I will be using the simplest method without any heuristics.

4. Methods

For this paper an experiment on PSO was conducted with three functions that are widely used in research on evolutionary algorithms (see Richards, Ventura[7] - also for graphical views on two-dimensional landscapes- and Angeline [2]). They are introduced below. The experiment tested geographical neighbourhood as introduced in section 3 against a fixed neighbourhood with the ring topology mentioned in section 2. Both used a neighbourhood of two neighbours to find their *lbest*. See section 4.2 for details on the experiment setup.
The first hypothesis is that a PSO search with fixed neighbourhoods will gradually turn into a search with geographical neighbourhood. To test this, two values were measured: The average distance a particle has to its two neighbours (“closeness”) and the average rank its neighbours have on a scale that ranks all particles with respect to the Euclidean distance to the particle in question (“georank”).
The second hypothesis is that geographical neighbourhood will lead to a more exploitative behaviour in the beginning of the search. To test for this, the fitness of the swarms was measured in the classical way: The average of the personal bests of all particles was recorded as well as the global best of the whole swarm.
The experiment also measured the variance of the data. So at every 100 iterations during the run the variance in the positions of the swarms was measured. Variance is a statistical method to measure how alike the entries in a vector are (the vector in this case would be the values of one dimension over all particles in the swarm). It sums over the squares of the distances between the entries and the mean of all entries of that vector and then computes the mean of that sum. I was interested in the variance on each dimension to measure how far spread the population is. So, for n particles, formula (3) describes the algorithm.

$$var(swarm) = \Sigma_{d=1}^{dimensions}(\Sigma_p^n((p_d - (\Sigma_p^n/n))^2)/n-1)/dimensions$$

Formula (3) The variance measure for a particle swarm

Of course, for each trial, the mean variance of all 30 swarms was taken into account.

4.1 Test functions

This subsection introduces the test functions that were used in the experiments. All of them were treated as minimizing problems.

Sphere The sphere function maximizes the absolute value on each dimension. It is therefore not a difficult problem to solve, but a good way to test the optimizing ability of an algorithm.

$$f(x) = \Sigma_{i=1}^n x_i^2$$

Formula (4) The sphere function

Rastrigin This is a function that has a lot of local optima that might deceive the algorithm.

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

Formula (5) The rastrigin function

Griewank The griewank function adds a lot of noise that can be deceiving the algorithm to exploit local optima.

$$f(x) = 1/4000\,\Sigma_{i=1}^n(x_i - 100)^2 - \Pi_{i=1}^n cos((x_i - 100)/\sqrt{i}) + 1$$

Formula (6) The griewank function

4.2 Setup

The experiment used a swarm size of 75 and a dimensionality of 10. The swarms were allowed to search for 300 iterations with the sphere and the rastrigin function and for 500 iterations with the griewank function (according to Angeline [2]). The results were averaged over 30 trials on each problem. The randomizer was the Mersenne Twister of Python 2.4. The particles values were initialized in the range of -15 to 15 for each problem. Both the cognitive and the social weight were set to 1.5. Finally, the values for the velocity were restricted to be within the range of -4 to 4 and the algorithms activity was also cooled down by gradually decreasing the momentum weight from 1.0 to 0.4 over the iterations.

5. Results

Below are the resulting graphs for each function. The left picture shows the fitness development. On the right side, the graphs describe the “closeness” and the “georank” for the two neighbourhood methods. Note that the y-axis on the fitness plots is logarithmic.

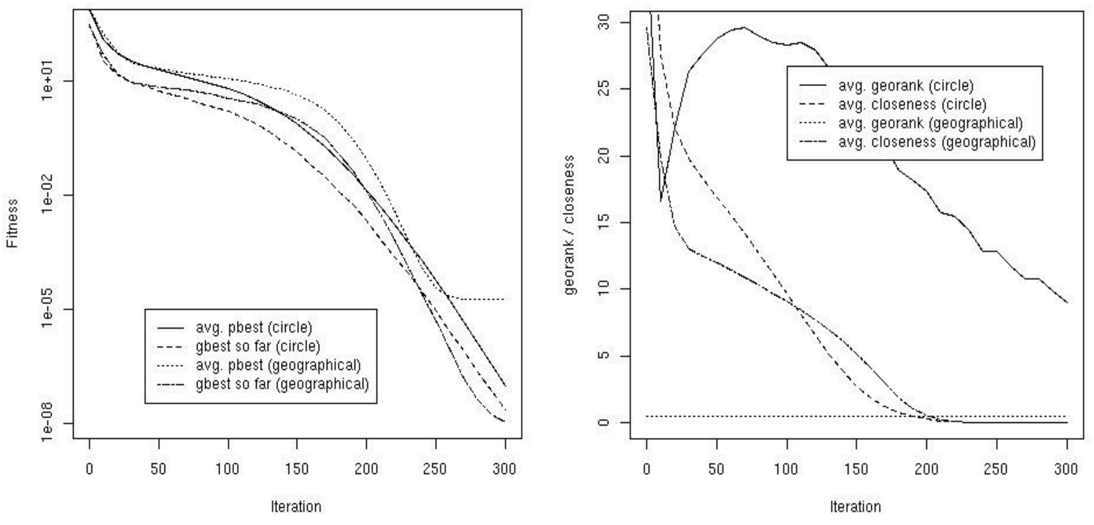


Fig. 1 Fitness and geographical data for the sphere function

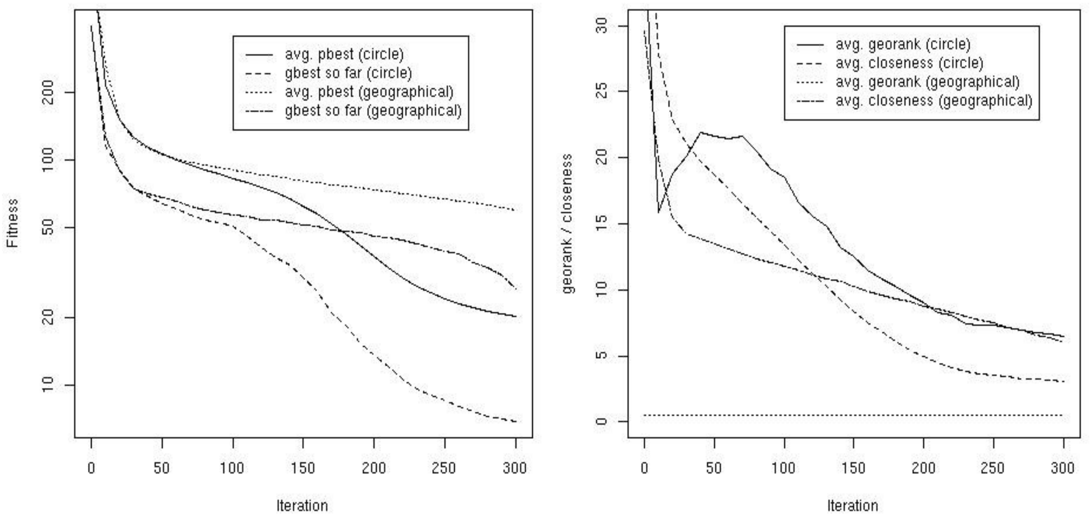


Fig. 2 Fitness and geographical data for the rastrigin function

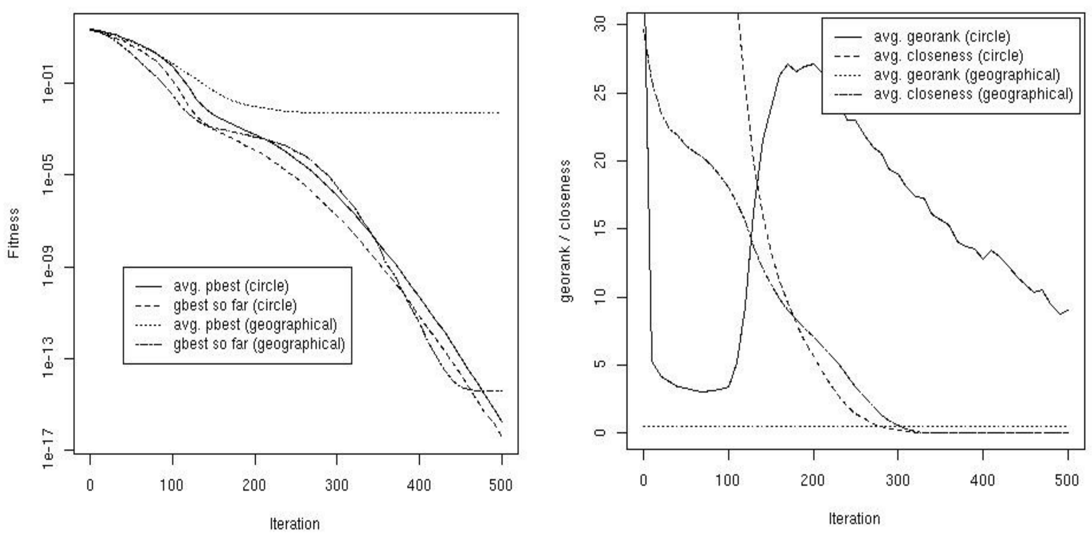


Fig. 3 Fitness and geographical data for the griewank function

5.1 Geometrical discussion

For the discussion of the geometric properties we are concerned with the graphs on the right of figure 1 through 3.
With all functions, the closeness of a particle to its neighbours declines and converges after about 200 to 300 iterations. As could be expected, geographical neighbourhood reaches much smaller distances between neighbours much faster. This is due to the fact that fixed neighbours are distributed everywhere over the search space when the algorithm starts. Geographical neighbourhood, on the other hand, is defined dynamically on the very basis of closeness. Nevertheless, the swarm with fixed neighbourhood will reach the same level of closeness at the point of convergence with the sphere and the griewank function (and even lower values shortly in between).
The georank measure is always 0.5 for the geographical neighbourhood. This could be expected, because as the closest neighbour has georank 0, the two closest neighbours have an average georank of (0+1)/2 = 0.5. For fixed neighbourhood, the results are more interesting: we observe two phases. The first starts at around 37.5 (or 75/2, half of the swarm size), which we would expect from a random initial distribution. We observe a decline in the very beginning of the run, followed by an increase to somewhere between 20 and 30. From there, the second phase lets the georank measure decrease again to values at around 10.
If we regard the two observed phases in the georank measure as exploration and exploitation, the “bump” makes sense: In the beginning, particles in the swarm with fixed neighbourhood orient towards their neighbours that are distributed over the whole search space. This is the exploration phase. In this phase, the particles build up much higher velocities than particles in the swarm with geographical neighbourhood. See formula (1): The next velocity is socially influenced by the distance between a particle and its best neighbour. High distances lead to high velocities. So the disposition to travel with a high velocity can be read from the closeness graph (we can also explain the higher slope for the closeness graph with fixed neighbourhood like this: High velocities lead to a faster decrease in distances).
When the swarm has found together, the georank measure will increase again, for particles move at relatively high velocities within a small space. The second phase is the exploitation phase. Velocities decrease for two reasons: The closeness measure of the particles decreases as does the momentum weight. In the exploration phase the swarm found together. Now it's the neighbours that find each other by exploiting the

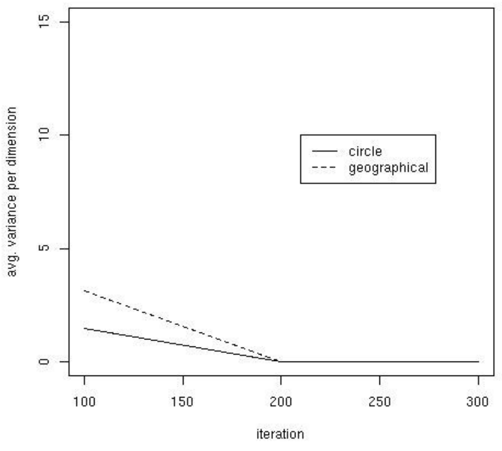
space between them.
The swarm with geographical neighbourhood, however, has not had a long exploration phase. Neighbours start finding each other from the start with no need for high velocities to explore much of space.

5.2 Fitness discussion

We are now concentrating on the left graphs of figure 1 through 3, each plotting the global best and the average local best at each iteration. We observe that the swarm with geographical neighbourhood converges on higher fitness values as the swarm with fixed neighbourhood is able to reach. At least this is the case for personal bests. The global best values for the sphere and griewank function are competitive, but within the last iterations, we observe convergence. For the rastrigin function, we don't see convergence, but global best values that are not competitive with fixed neighbourhood.
The premature convergence for this swarm can be explained by the geographical discussion in section 5.1. Because it does not pick up high velocities, the swarm with geographical neighbourhood is exploiting almost from the beginning on. The decreasing momentum weight leads to further decrease in velocities. Without high velocity, the probability of finding good solutions is very low.

5.3 Variance discussion

Find the graphs describing the variance at iterations 100, 200 and 300 (and 400 and 500 for griewank) according to formula (3) below.



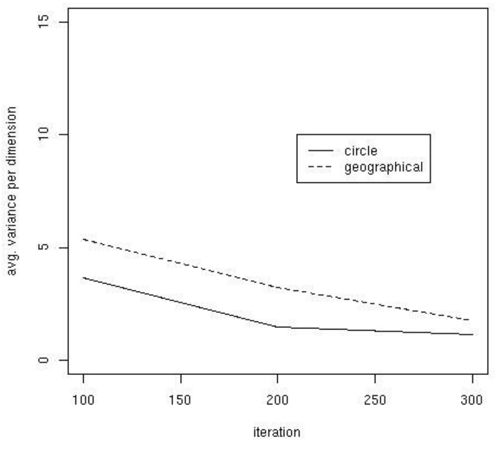


Fig. 4 Variation with sphere function **Fig. 5 Variation with rastrigin function**

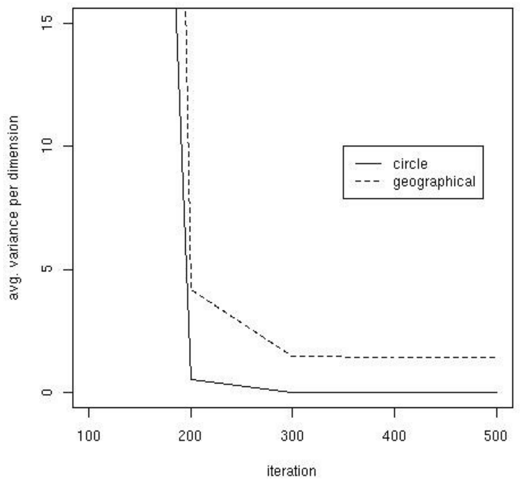


Fig. 6 Variation with griewank function

With every function, we see that the variance in swarms with geographical neighbourhood decreases slower, which matches the expectation. Slower particles will decrease variance slower. With the sphere function, swarms with both kinds of neighbourhood reach an equal level of variance after 200 iterations. With the rastrigin function, geographical neighbourhood is near the level of fixed neighbourhood after 300 iterations. With the griewank function, we observe that the variance of fixed neighbourhood stays on a higher level than fixed neighbourhood. Also, the variance with both topologies is considerably high within the first 200 iterations.
While we note that in most cases, geographical neighbourhood loses almost as much variance on the long run as fixed neighbourhood, the differences in variance on the way there can be quite large. For instance, at 200 generations, with the rastrigin function we see about 100% more variance in geographical neighbourhood and with the griewank function about 500%. This is not saying much about the significance of that variance in terms of adaptability of the swarms to changing conditions (see Carlisle, Dosier [3]), but it might be a starting point for further investigation on dynamically changing environments and geographical neighbourhood.

I will now reconsider the two hypothesis made in the introduction. The first hypothesis was that a PSO search with fixed neighbourhoods will gradually turn into a search with geographical neighbourhood. The closeness measure shows that this is indeed true. The distances between neighbours in both topologies are hard to distinguish in the end of the runs. In between fixed neighbourhood performs even better on that scale.
I also hypothesized that for geographical neighbourhood, the exploration/exploitation - trade- off will tend to exploitation. This also seems to be true, and the implications in the fitness plots are dramatic.

Thus, the very nature of swarms with a fixed neighbourhood topology seems to be that they go through two phases: They take advantage of an initial random distribution to explore at high velocity and then they turn to exploitation when they found together and velocities become lower. In terms of achieving explorative behaviour, fixed neighbourhood topologies have randomness in their favour. Geographical neighbourhood topologies, on the other hand, seem to miss a comparable mechanism to explore a search space.

6. Conclusions & Future Work

In this paper I introduced geographical neighbourhood for Particle Swarm Optimization. I introduced a simple algorithm to compute Euclidean distance and discussed its complexity as well as possible further improvements. I conducted a standard PSO experiment comparing geographical and fixed neighbourhood. The results showed that swarms with geographical neighbourhood do not reach comparable fitness values as swarms with fixed neighbourhood do. Geographical measures such as closeness of neighbours, geometrical neighbourhood rank and variance of positions allowed learning about the behaviour of both geographical and fixed topologies. The hypothesis that swarms with fixed topologies explore because of random distribution and then exploit because of emerged geographical neighbourhood seems to hold.
The observation that swarms with geographical neighbourhood retain higher variances of positions might be interesting when the environment is not static, but changes gradually over time. Future work could explore the performance of geographical neighbourhood on such environments.

References

[1] Angeline, P. 1998: *Evolutionary Optimization Versus Particle Swarm Optimization* in *Proceedings of the 7th International Conference on Evolutionary Programming VII***,** pp 601- 610
[2] Angeline, P. 1998: *Using Selection To Improve Particle Swarm Optimization* in *Evolutionary Computation Proceedings*, pp 84- 89
[3] Carlisle, Dosier 2000: *Adapting Particle Swarm Optimization To Dynamic Environments* in *Proceedings of International Conference on Artificial Intelligence*
[4] Eberhart, R. and Shi, Y. 1998: *Comparison between Genetic Algorithms and Particle Swarm Optimization* in *Proceedings of the 7th International Conference on Evolutionary Programming VII*, pp 611- 616
[5] Kennedy, 1997: *The Particle Swarm: Social Adaptation Of Knowledge* in *IEEE International Conference on Evolutionary Computation, 1997*, pp 303- 308
[6] Kennedy, Mende*s* 2002*: Population Structure and Particle Swarm Performance* in *CEC '02. Proceedings of the 2002 Congress on Evolutionary Computation, 2002*, pp 1671- 1676
[7] Richards M., Ventura D. 2002: *Dynamic Sociometry in Particle Swarm Optimization* in *International Conference on Computational Intelligence and Natural Computing, North Carolina, 2003*
[8] Suganthan 1999: *Particle Swarm Optimizer With Neighbourhood Operator* in *CEC 99. Proceedings of the 1999 Congress on Evolutionary Computation, 1999,* pp 1557- 1962