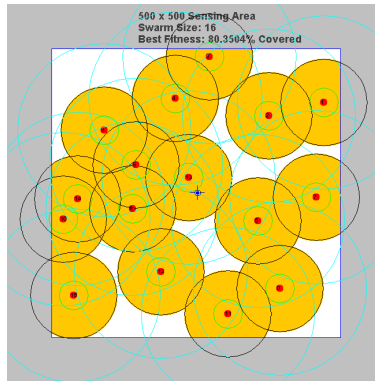


# Optimum Coverage of the Autonomous Sensor Swarm: Sensor PSO

Matt Stabeler

School of Computer Science and Informatics University College Dublin



## Abstract

A common problem of autonomous sensor networks is the placement of sensor nodes to give adequate coverage of a sensing area. This study applies particle swarm optimisation techniques in the form of Sensor PSO to evolve good solutions to the coverage problem. 4 experiments were performed evaluating Sensor PSO against random placement of sensors, a further 2 were performed using altered variables of the Sensor PSO algorithm. The results showed that Sensor PSO was effective in producing better results, and found that changing certain variables increases the quality of the results. The conclusion is that particle swarm optimisation can be helpful in positioning sensors versus random placement of sensors.

## 1 Introduction

In the realm of autonomous sensor systems a common issue [7] is the positioning of sensor nodes efficiently to reduce the cost and effort of deployment, but maximise the connectivity and coverage of the network. It is also very relevant to the automatic positioning of mobile sensors/devices for example, BOIDS [4] or Intelligent Traffic Cones [8].

There are a number of studies and suggestions on how to solve the problem of coverage and organisation of wireless sensor nodes [13][12][7], and some [6][9] have suggested PSO as a solution.

A sensor can have a number characteristics, they are able to communicate with a central node, or other sensors, often using technologies such as Bluetooth, WiFi, Zigbee, RFID, Infra-red and other communications technologies, this gives them a limited communications range. The main aspect of their operation is their sensing abilities, this ranges from gyrometers to heat sensors to GPS signal sensors to optical sensing (photographic) to proximity sensing and motion detection to database monitoring agents. These sensing abilities also have a finite range, and can be considered to be their sensing range. Sensors are often deployed in swarms, numbering a few [2] to many hundred or thousands [11]. In order for these nodes to communicate and sense effectively, it is necessary to optimise their positioning within the sensing area, and in fact sensors may well have their own autonomous or manual mobility, to this end particle swarm optimisation techniques may help to autonomously organise these sensors efficiently, without the need for manual control. Power conservation is also a constraint when considering wireless sensor networks, as often [13] sensors have limited battery life, such as the Shimmer [2], which has to be recharged when the battery is low.

This study covers the optimisation of the position of sensors with regards to their sensing range, however it does not cover complex networking calculations, and uses simple, perhaps naive calculations to account for communications range.

Results, source code and a running version of SensorPSO is available online [14] for testing and verification.

## 2 Hypothesis

A group of sensors can be optimised using Sensor PSO to form a more efficient sensing coverage of a given area, than with random placing of sensors.

## 3 Methodology

Six experiments were performed to test the effectiveness of using particle swarm optimisation to find the most efficient placing of sensors within a simple sensing area. Three of these experiments involved randomly placing sensors, as a measure of particle swarms effectiveness.

These experiments applied optimisation by maximisation, to find the best coverage using a given set of rules. To test this hypothesis a software platform was adapted to implement a system upon which to run multiple swarm optimisations in a batched manner. This section discusses the experiment design, setup and execution.

### 3.1 Platform

In order to reproduce and test a solution to the problem, a software platform was sought that would be simple to use, but at the same time be able to demonstrate the solution reliably and in a manner that could easily be understood, ideally visually as in real-world situations, it may be easier to place sensors from a visual 'map' rather than using only a set of co-ordinates. Three candidate solutions were found and evaluated.

Firstly, ECJ [1] was evaluated for its effectiveness for solving this solution. It is described on its webpage as follows:

ECJ is a research EC system written in Java. It was designed to be highly flexible, with nearly all classes (and all of their settings) dynamically determined at runtime by a user-provided parameter file. All structures in the system are arranged to be easily modifiable

ECJ's nature is to evolve a solution and present it textually as a file, and/or to the console. It is a very powerful system that allows a very large number of configurations, however there is no easy to use visualisation feature.

The second pre-built system evaluated was JSwarm [3], which is described simply by its website:

JSwarm-PSO is a Particle swarm optimization package written in Java.

JSwarm-PSO was found to be an interesting implementation for PSO, it implemented capability for visualising the PSO process for one example problem. However it was found to be difficult to adapt to other problems whilst maintaining a useful visual interface.

Finally, a program called PSopt[5] was tested, it is described by the website as:

A program to demonstrate the optimisation process of particle swarm optimisation.

This program was found to be useful for visualising results, and also simple to configure and adapt.

After evaluating these three candidate software platforms, PSopt which is released under the GNU General Public License, was chosen due for this study due to its capability for visualisation and simplicity of adaption.

### 3.2 Rules

The problem of optimising sensor placement requires a special set of rules to calculate the fitness of the particle and determine the behaviour of the particle swarm. This section discusses the rules and properties that were identified that affect the particle sensor swarm, it also describes the platform implementation of these rules.

The object of the sensor swarm is to cover as much sensing area as possible, however, sensors need some overlap, to cover an area fully [10], to that end the following Rule 1 restricts the number of sensors covering the same sensing area. However, sensors should not cover the same sensing area, and should tend not to be too close to each other, as this is inefficient. In a real-world sensor deployment, data may be required to report back to a central node [11], or it may just be to cover a central point, networking calculations are beyond the scope of this project, and for this reason, it was assumed that a range of twice an individual sensors communications range would be sufficient for all sensors to network to a central node. If the sensing area is a (symmetrical) fixed space (as with this study) we would like to make to swarm tend towards the centre, as a grouping point, covered by rule 4. This experiment has a fixed area to be sensed, outside of which a particle has no benefit (rule 5). This special problem relates directly to a real life sensor networks, which ideally need to form efficient networking and sensing structures, therefore the swarm should naturally stop or slow when it finds a good pattern. The following rules try to account for all of these constraints:

- Rule 1: A particle must not be in sensing range of more than a few other particles (a proportion of the swarm).
- Rule 2: A particle must not be too close to any other sensor.
- Rule 3: The swarm must be in communications range of the global best.
- Rule 4: The swarm should tend towards the centre of the sensing area.
- Rule 5: The particle should stay within the bounding box wherever possible
- Rule 6: The swarm should attempt to settle into the most efficient pattern

### 3.3 Implementation

The software system was adapted to use these special set of rules to process the swarm calculations; Sensor PSO. Each particle within the swarm has certain properties that help the algorithm to calculate its behaviour:

- `communications_range` in pixels is the radius in which the sensor can communicate
- `sensing_range` in pixels is the radius in which the sensor can sense
- `min_peer_distance` in pixels determines the range in which another sensor would be too close.
- `min_sensors_in_comms_range` is the minimum number of sensors that should be within communications range
- `ratio_of_swarm_sensed` is the ratio of how much of the swarm can be within sensing range (e.g. a value of 3 is one third).
- `max_too_close` is the maximum number of sensors that should be within `min_peer_distance`

**Fitness** Two fitness functions were developed to help evaluate the fitness of both the individual particle and the swarm as a whole. The fitness of the swarm is calculated as the percentage of the area covered by the swarm, this is done by polling each pixel in the bounding box, to see whether it is within sensing range of a particle. This is used to identify the best pattern of sensors during the run of the algorithm; when the swarms fitness is better than that of the previous best coverage, the fitness and the positions of each individual particle is recorded. The individual fitness of each sensor is calculated by testing for the rules determined previously. Firstly, a general calculation is made to increase the fitness value towards the centre of the grid:

$$fitness = -4 * ((px^2) + (py^2))$$

where `(px, py)` is the co-ordinate of the particle in normalised space from `(-1.0, -1.0)` to `(1.0, 1.0)`. This function increases the fitness towards a value of `0.0`. The particle is then tested to see whether it is near to the centre of the grid; the arbatory figure of twice the communications range is used as the test of near distance (as discussed in section 3.2). If the sensor is not near, then the fitness value is decreased by the value `1000.0`. Next; the particle is tested to determine if it is currently within the bounding box, if this function returns false the fitness is reduced by a further value of `1000.0`. If the particle is found not to be on the screen, the fitness is reduced by the value of `10.0`. If the particle is found to have more than 4 sensors within its sensing range, or more than 1 particle is too close, then the sensor fitness is set to the minimum possible `Double` value.

**Update** This function is applied to each particle in turn during the update phase of the algorithm. Firstly a calculation is made to test if the particle is in a good position for it to stay in the same place, by testing a number of aspects if its position; If the sensor is in near the centre of the grid (as discussed above), on the screen, in the bounding box, has at least one other sensor within communications range no more than one third of the swarm within sensing range and no other sensor too close, then the particles position is updated using a randomising function to give it an offset of between `0.001` and `1.0` for each axis within the normalised space using the calculation `Double offset = (double) 1 / (r.nextInt(750) + 1)`; and randomly assigned negative or positive value. This has the effect of stopping the particle near its current location.

If the stopping criteria above is not met then the particle is tested to see if it has started to converge with other swarm members; if the number of particles in sensing range is more than one third of the swarm, then a function is applied to move the particle a few (random in any direction) pixels away, so that it no longer occupies the same space as other particles.

The default action of the update function is to tend particles towards the centre, so if neither of the the above criteria are met, then the particle is updated using the following calculation:

$$\begin{aligned}x &= x + (\alpha * vx + \beta 1 * (bx) + \beta 2 * (gx - x)) \\y &= y + (\alpha * vy + \beta 1 * (by) + \beta 2 * (gy - y))\end{aligned}$$

Where `x` and `y` are the co-ordinates of the particles, `vx` and `vy` represent the current velocity of the particle, `bx` and `by` represent the locally best position found so far by the particle and `gx` and `gy` is the best position found globally. The `α` value serves to decelerate the swarm over time using the `decay` attribute of the swarm, the `β1` and `β2` values are random values that allow the velocity updates to veer away from the centroid between the local best and the global best.

**Output** When each run completes, the best recorded pattern of the swarm is re-initialized, a screenshot taken as shown in Figure 1, and a text file created with the best history of the whole run, including particle positions at the end of each update cycle and fitness of the swarm at that cycle. The screenshot shows the individual particles/sensors (numbered red dots), the ranges (concentric circles), from innermost to outermost; green is minimum peer distance, black is the sensing range and cyan is the communications range. The sensing range is filled to easily show the what area is covered (solid yellow/orange). The bounding box is the inner blue lined, white filled square, and the remaining viewport (outer box) is light grey. The blue circle and cross (centre) is the global best position, and the smaller blue crosses (if visible) are the individual best positions of the each memeber of the swarm.

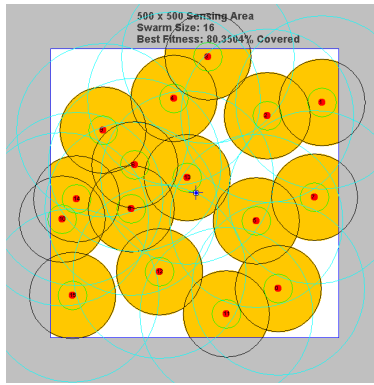


Figure 1: Example of output

### 3.4 Experiment Setup

Each experiment used a 500 by 500 pixel square (bounding box) as the area to be 'sensed'. This was enclosed in the viewable area of the program (viewport) of approximately 650 by 650 pixels.

Particles (sensors) were allowed to leave the bounding box and the viewport and exist outside of the viewable area. However, in each instance, the best global position (**gBest**) was only allowed to be placed within the bounding box. The centre of the viewport was considered to be the central point (Point 0,0) of normalised (between -1.0 and 1.0) 2D space.

The approximate sensing coverage of a sensor with a sensing range of 75 pixels is 7 percent, a swarm population of 16 was chosen as which has a physical maximum of 112 percent coverage. During development, this number of particles was found to be a good number of sensors to deploy. Smaller numbers of sensors say 5, would be able to achieve their maximum coverage, without converging into a swarm, this would cause unreliable results, by the same token, using a high number of sensors, say 50 may 'blot out' the area, giving consistently high coverage, but with a poor distribution and lack of settlement.

**Round 1: Optimum Coverage** In every instance, 16 sensors were added to the viewport randomly, and initialised with the following radii; sensing range of 75 pixels, communications range of 150 pixels and a minimum peer distance of 25 pixels. Particles were also initialized with an acceleration factor of 1, and a decay factor of 0.01, the effect of which is to slowly reduce the velocity of the swarm (but not in the case of random placement).

Experiments 1 (Sensor PSO) and 3 (Random Placement) were initialised to run for 1000 epochs (update cycles) each, this allowed the swarms to explore fully, the area and hopefully produce the best results.

Experiments 2 (Sensor PSO) and 4 (Random Placement) were given 100 epochs in which to find the best solution, this challenged the swarms to quickly find a solution, akin to a real-world example of mobile sensors.

The list below shows the setup of all aspects of the experiments:

- Exp 1 - Sensor PSO 1000 Epochs, 16 Sensors, acceleration 1, deceleration 1, decay 0.01, Sensing range 75, Comms Range 150, Min Peer distance 25, Min Comms in range 1, Max Peers close 0, Max in Sense range 1 in 3
- Exp 2 - Sensor PSO 100 Epochs, 16 Sensors, acceleration 1, deceleration 1, decay 0.01, Sensing range 75, Comms Range 150, Min Peer distance 25, Min Comms in range 1, Max Peers close 0, Max in Sense range 1 in 3
- Exp 3 - Random Placement 1000 Epochs, 16 Sensors, acceleration 1, deceleration 1, decay 0.01, Sensing range 75, Comms Range 150, Min Peer distance 25, Min Comms in range 1, Max Peers close 0, Max in Sense range 1 in 3
- Exp 4 - Random Placement 100 Epochs, 16 Sensors, acceleration 1, deceleration 1, decay 0.01, Sensing range 75, Comms Range 150, Min Peer distance 25, Min Comms in range 1, Max Peers close 0, Max in Sense range 1 in 3

**Round 2: Tweaks** In round two of the experiments, 16 sensors were again added to the viewport randomly; but two variables were changed, to measure their affect upon the swarms performance. The sensing radius and communication radius remained fixed at 75 pixels and 150 pixels respectively, however the minimum peer distance was increased to 50 pixels from 25. The minimum of 1 sensor in communications range was upheld as well as the maximum number of sensors too close remained 0. The ratio of swarm particles that were allowed to be within sensing range was reduced to one fifth from one third. Listed below are the setup parameters of these experiments.

- Exp 5 - Sensor PSO 1000 Epochs, 16 Sensors, acceleration 1, deceleration 1, decay 0.01, Sensing range 75, Comms Range 150, **Min Peer distance 50**, Min Comms in range 1, Max Peers close 0, **Max in Sense range 1 in 5**

- Exp 6 - Sensor PSO 100 Epochs, 16 Sensors, acceleration 1, deceleration 1, decay 0.01, Sensing range 75, Comms Range 150, **Min Peer distance 50**, Min Comms in range 1, Max Peers close 0, **Max in Sense range 1 in 5**

## 4 Results

The results presented here were generated as described above in a batched manner, and can be found online [14] along with screenshots of all best of runs. The graphs in figures 2 to 9 show the results of the best solution found for each run, in the case of experiment 1, an error was made in the number of runs performed, resulting in 10 less runs. Figures 6, 7 and 10 show scatter plots of epochs at which the best results were found.

### 4.1 Round 1 Results

The overall average fitness for experiments 1 and 2, representing the first Sensor PSO algorithm is 73.86, approximately 1.5 percent better than the average for the random placement algorithm at 72.59. The highest scoring individual run was found in Experiment 1 at run 37, epoch 71, which scored 80.35 percent fitness, this was followed by Experiment 3, run 26 epoch 951, at 77.77 percent, closely followed by Experiment 2, run 34 epoch 74, at 77.61 percent and finally Experiment 4, run 5 epoch 37, at 76.29 percent.

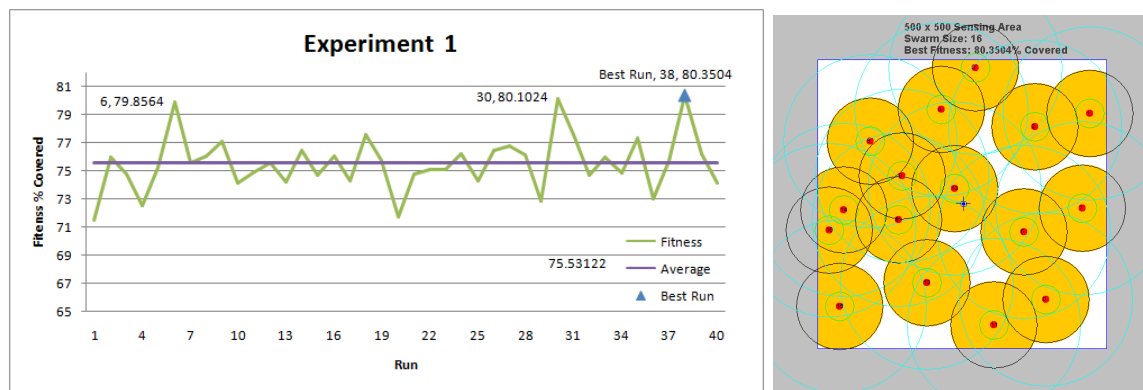


Figure 2: Experiment 1, Best fitness from each run and Best solution

Experiment 1 had high scores in runs 5, 27(best) and 29 at 79.85, 80.35 and 80.10 percent respectively; illustrated in Figure 2. Experiment 2 also demonstrated peaks in performance, and had 4 peaks at run 17, 35 (best), 41 and 49 (see Figure 3).

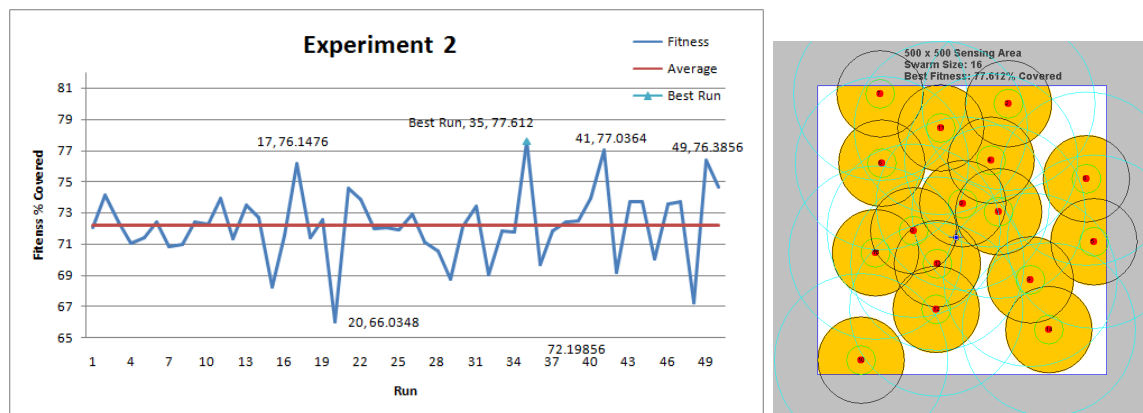


Figure 3: Experiment 2, Best fitness from each run and Best solution

The results for the random placement algorithm also show peaks of performance, experiment 3 has peaks in run 26 (best), 32 and 47. Also demonstrated are a number of low scores, notably in Experiment 2 which demonstrates the lowest of the first 4 experiments at run 19 with 66.03 percent coverage. Closely followed by Experiment 4 with a minimum of 66.67 at run 14.

Figures 6 shows the epochs at which the best results were found, over all of the runs using the Sensor PSO algorithm, the graph is normalised for the differing number of epochs, Experiment 1 scale on the right (having 1000 epochs) and Experiment 2 scale on the left (having 100 epochs). The Sensor PSO algorithm shows a slight

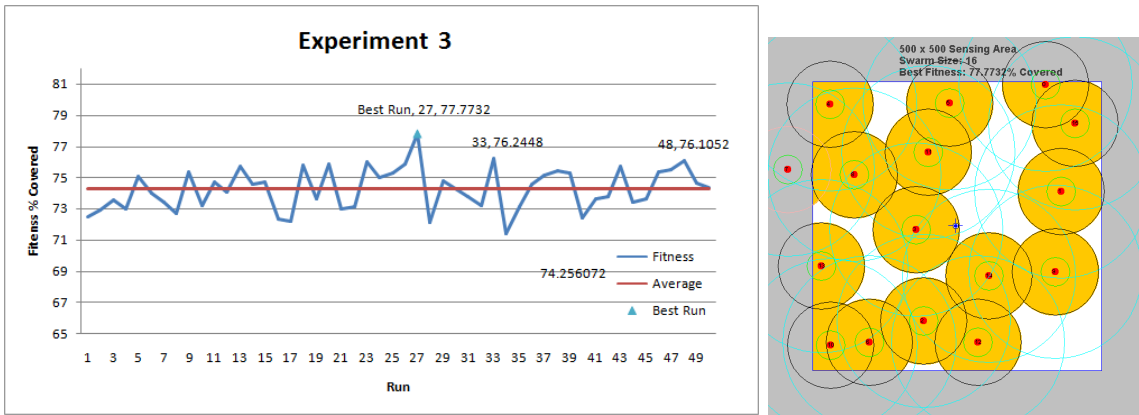


Figure 4: Experiment 3, Best fitness from each run and Best solution

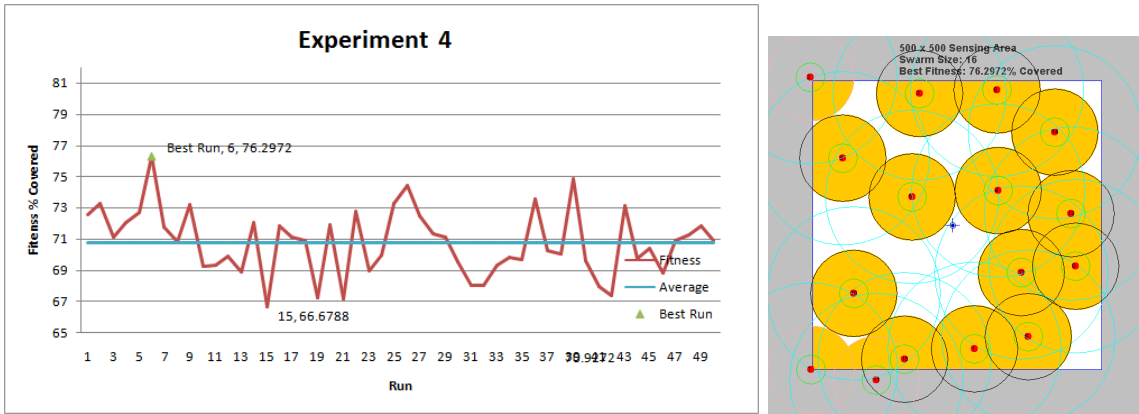


Figure 5: Experiment 4, Best fitness from each run and Best solution

trend for finding best positions early on in the sequence when the number of epochs is low (100), and for finding good positions later on in the sequence when the epochs are higher (1000).

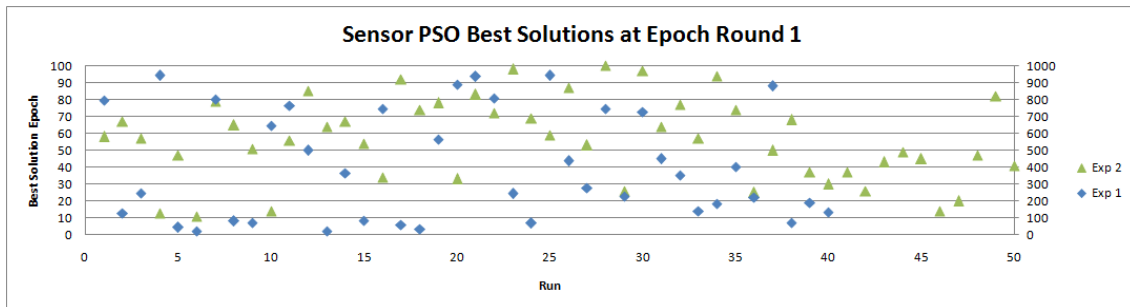


Figure 6: Scatter plot of epochs at which best solution was found for Sensor PSO

The random placement algorithm produces the scatter plot shown in Figure 7 for the epochs at which it found the best fitness, it again used a normalised scale with Experiment 3 to the right and Experiment 4 to the left. There appears to be no easily discernable trends in this data, apart from a random spread of points.

## 4.2 Round 2 Results

Round 2 of experiments produced results only for the Sensor PSO algorithm, Figure 8 shows the results from 20 runs of 1000 epochs in experiment 5, and clearly shows a number of high peaks at 1, 2, 4, 7, 11, 16, 18(best) and 19. There are areas of poorer performance, but none drop below 71 percent. This experiment produced the an overall highest result of 78.66.

Experiment 6 managed to achieve a best fitness of 80.37 percent at run 4, Epoch 39, which makes the average of these two runs 75.49, 1.5 points higher than experiments 1 and 2 and nearly 3 points higher than experiments

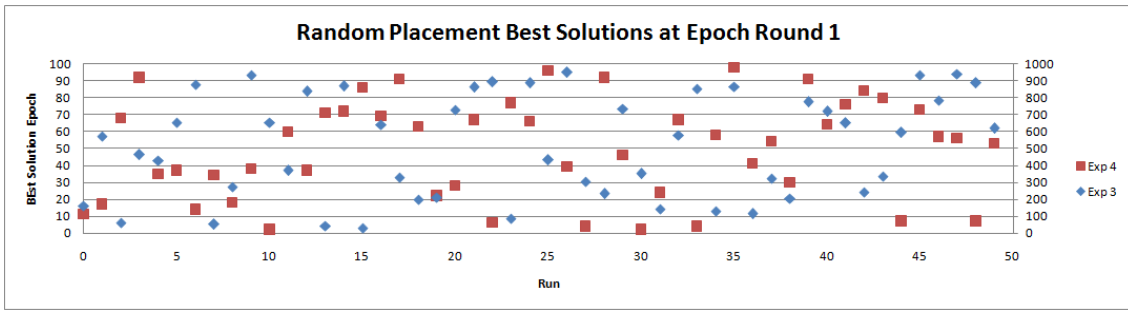


Figure 7: Scatter plot of epochs at which best solution was found for random placement

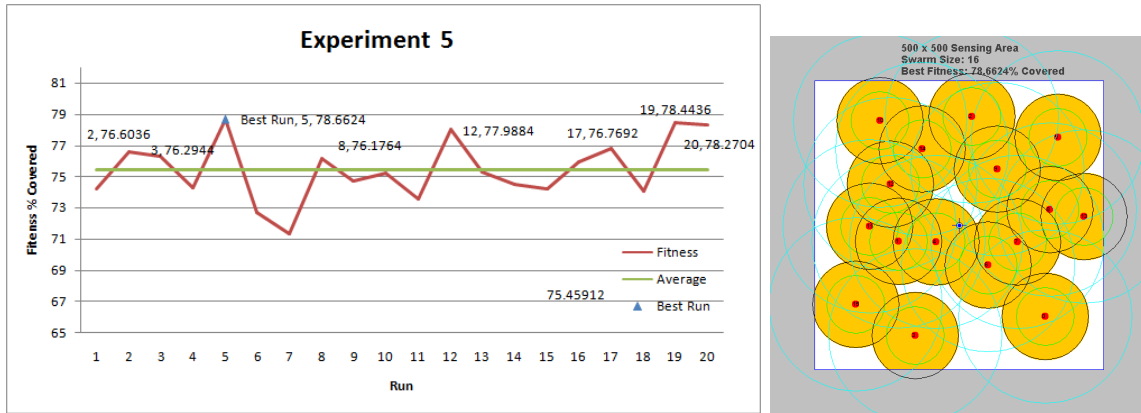


Figure 8: Experiment 5, Best fitness from each run and Best solution

3 and 4. This again shows a high number of peaks at runs 0, 8, 14, 15, 16, 17 (best run), 18 and 19.

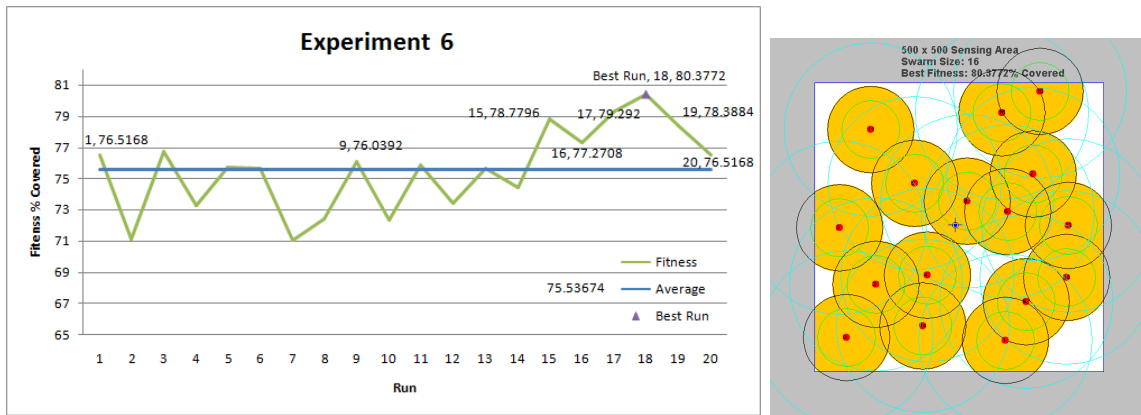


Figure 9: Experiment 6, Best fitness from each run and Best solution

The epochs at which the best fitness is found is shown in Figure 10 and shows a general trend for finding the best results early on in the cycle, however, only 20 runs were performed for both 100 and 1000 epochs, so if any trends were found in later runs of previous experiments, they could not be reliably compared here.

## 5 Conclusion

Of the 6 experiments performed, experiment 1 (Sensor PSO) and experiment 6 (Sensor PSO Round 2) produced the best fitness, however experiments 1 and 2 (Sensor PSO) had a larger deviation, and showed some poorer results than experiments in round 2. Experiments 3 and 4 (Random Placement) showed a trendless set of results, and performed randomly across runs.

Experiment 5 and 6 (Sensor PSO) showed the highest average score, and neither scored less than 71 percent fitness, compared to the lowest fitness scores of 66.03 percent and 66.67 percent in experiments 2 and 4.

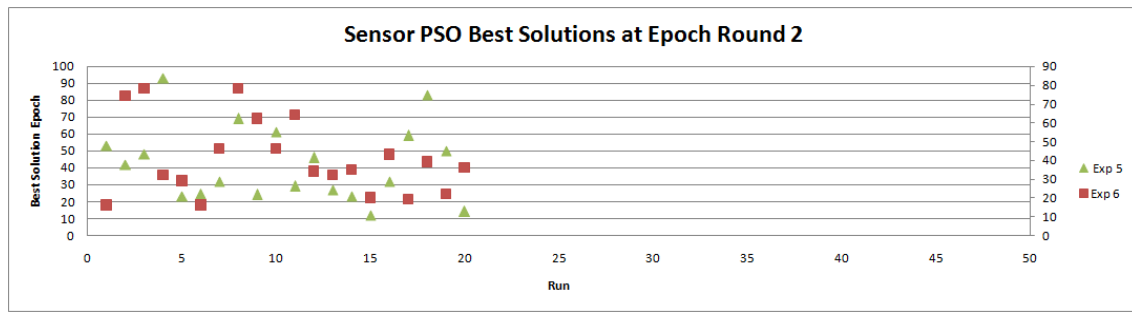


Figure 10: Scatter plot of epochs at which best solution was found for Sensor PSO at round 2

Experiments 5 and 6 also showed trend towards finding the best solution quickly.

Overall, the results show that using particle swarm optimisation is better than using random placement to optimise the coverage of sensors, however, using this algorithm it is only slightly more effective at finding a better pattern, but when the parameters for minimum peer distance is increased and number of sensors in sensing range are increased, the Sensor PSO algorithm performs better overall, finding a higher number of good quality solutions than random placement alone.

## 6 Discussion and Further Work

### 6.1 Discussion

It is perhaps naive to assume that this algorithm could apply to real-world situations, due to its simplistic view of some of the characteristics of a sensor, say, the fields of sensing, communications and movement may be limited and irregular, for example a communications radius is unlikely to be a perfect circle, and the movement of a sensors is likely to be slow, and energy in-efficient if moving as this algorithm moved its particles. However, this algorithm may be useful to implement a swarm that can run a simulation, find the best possible positions, and then move toward them.

### 6.2 Further Work

This short study was not able to cover all of the areas the author would have liked, and during the course of the study the following aspects were pondered.

**Optimum Parameters** This study has shown that the fitness of the swarm can be affected by adjusting parameters of its behaviour, a future work would be to find the most optimum of these settings, to suit different situations, for example a location sensing swarm may need more particles within sensing range, and a object detection swarm may need to be spread more thinly.

**Multiple global bests** If a sensor swarm is needed to cover two main points of interest, then it would be interesting to implement a swarm with two global best positions, to have the swarm converge to two different points of coverage, perhaps two separate swarms could achieve this.

**Different shaped bounding boxes** Similar to the multiple global best idea is having different shaped bounding boxes, circles, triangles etc. as areas within which sensors should sense. Combine this with a concept corridors and doorways, the particle swarm may be able to sense a more real-world layout of a room and building.

**Random sized sensing ranges - realworld** Hinted at in the discussion above, another factor that would enhance this study would be to implement different sized ranges for particles, perhaps representing different sensor types. This would test the algorithms ability to sort sensors into better configurations.

**Redundancy** Building in a redundancy calculation may help to define when particles are having no benefit to the swarm for example if a sensing area has too many sensors, how does the network decide which ones to turn off to conserve power.

## References

- [1] Ecj website. <http://cs.gmu.edu/eclab/projects/ecj/>. Last accessed 06 Nov 2007.
- [2] Hardware - tril centre. [http://www.trilcentre.com/technology\\_platform/hardware.568.460.html](http://www.trilcentre.com/technology_platform/hardware.568.460.html). Last accessed 20 Nov 2007.

- [3] Jswarm-pso website. <http://jswarm-pso.sourceforge.net/>. Last accessed 06 Nov 2007.
- [4] "boids (flocks, herds, and schools: a distributed behavioral model)". "Internet", "October" "2007".
- [5] Christian Borgelt. Christian borgelt's webpages, psopt website. <http://www.borgelt.net//psopt.html>. Last accessed 06 Nov 2007.
- [6] "Shixing Gao " "Ping Yuan " "Zhe Li" "Chunlin Ji", "Yangyang Zhang ". "particle swarm optimization for mobile ad hoc networks clustering". In "*Networking, Sensing and Control, 2004 IEEE International Conference on*", volume 1, pages 372–375, "Sch. of Inf. Sci. Eng., Northeastern Univ., Shen Yang, China", March 2004.
- [7] Deborah Estrin, Ramesh Govindan, John Heidemann, and Satish Kumar. Next century challenges: Scalable coordination in sensor networks. In *Proceedings of the ACM/IEEE International Conference on Mobile Computing and Networking*, pages 263–270. ACM, August 1999.
- [8] Shane M. Farritor and Steve Goddard. Intelligent highway safety markers. *IEEE Intelligent Systems*, 19(6):8–11, 2004.
- [9] B. Anthony Kadrovach and Gary B. Lamont. A particle swarm model for swarm-based networked sensor systems. In *SAC '02: Proceedings of the 2002 ACM symposium on Applied computing*, pages 918–924, New York, NY, USA, 2002. ACM Press.
- [10] R.; O'Hare G.M.P.; Ruzzelli A. Marsh, D.; Tynan. The effects of deployment irregularity on coverage in wireless sensor networks. *Intelligent Sensors, Sensor Networks and Information Processing Conference, 2005. Proceedings of the 2005 International Conference on*, pages 13–18, 5-8 Dec. 2005.
- [11] Stephan Olariu, Mohamed Eltoweissy, and Mohamed Younis. Answer: Autonomous networked sensor system. *J. Parallel Distrib. Comput.*, 67(1):111–124, 2007.
- [12] Kafil M. Razeeb, Stephen Bellis, Brendan O'Flynn, John Barton, Kieran Delaney, and Cian O'Mathuna. A hybrid network of autonomous sensor nodes. In *EUSAI '04: Proceedings of the 2nd European Union symposium on Ambient intelligence*, pages 69–70, New York, NY, USA, 2004. ACM Press.
- [13] J. Ailawadhi V. Pottie G.J. " "Sohrabi, K. Gao. "protocols for self-organization of a wireless sensor network". "*Personal Communications, IEEE [see also IEEE Wireless Communications]*", 7(5):16–27, oct 2000.
- [14] Matt Stabeler. Sensor pso. <http://mattstabeler.co.uk/projects/sensorpso/>.