Earthquake Classifying Neural Networks Trained with Random Dynamic Neighborhood PSOs

Arvind S. Mohais
Dept. Math. & Comp. Sci.
University of the West Indies
St. Augustine, Trinidad
amohais@fsa.uwi.tt

Rosemarie Mohais
Seismic Research Unit
University of the West Indies
St. Augustine, Trinidad
rmohais@uwiseismic.com

Christopher Ward
Dept. Math. & Comp. Sci.
University of the West Indies
St. Augustine, Trinidad
cward@fsa.uwi.tt

Christian Posthoff
Dept. Math. & Comp. Sci.
University of the West Indies
St. Augustine, Trinidad
cposthoff@fsa.uwi.tt

ABSTRACT

This paper investigates the use of Random Dynamic Neighborhoods in Particle Swarm Optimization (PSO) for the purpose of training fixed-architecture neural networks to classify a real-world data set of seismological data. Instead of the ring or fully-connected neighborhoods that are typically used with PSOs, or even more complex graph structures, this work uses directed graphs that are randomly generated using size and uniform out-degree as parameters. Furthermore, the graphs are subjected to dynamism during the course of a run, thereby allowing for varying information exchange patterns. Neighborhood re-structuring is applied with a linearly decreasing probability at each iteration. Several experimental configurations are tested on a training portion of the data set, and are ranked according to their abilities to generalize over the entire set. Comparisons are performed with standard PSOs as well as several static non-random neighborhoods.

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Applications and Expert Systems

General Terms

Algorithms

Keywords

Particle Swarm Optimization, Neural Networks, Neighborhood Configurations, Dynamic Neighborhoods, Earthquake Classification.

GECCO'07 July 7–11, 2007, London, England, United Kingdom

1. INTRODUCTION

Particle Swarm Optimization (PSO) [5] is a robust evolutionary algorithm that continues to be studied both for its intrinsic properties as well for applications. The basic PSO algorithm has undergone many modifications and one area of research that has accelerated recently is neighborhood structures [7, 11, 9]. This paper uses the idea of randomly generated neighborhoods that are dynamically altered during the run of the algorithm [13, 12] to solve to the real-world problems of classifying seismological strong ground motion data.

Earthquake activity is particularly important in Trinidad and Tobago where there are critical economic assets such as liquid natural gas (LNG) plants that are subject to a significant hazard level [15]. Damage to structures caused by earthquakes is most often the result of strong ground motion [17], and monitoring this type of effect is crucial. A large set of strong motion data for the Trinidad and Tobago region has been procured for use in this research from the Seismic Research Unit. The goal was to develop a neural network that correctly classifies each data entry as either an earthquake or not.

PSOs were used to evolve weights for several fixed neural network (NN) architectures. Each particle represented a set of weights for a particular architecture and evaluation was done by applying the resulting NN on half of the data set. After all PSO configurations were run, a final ranking of the evolved NNs was done by applying them to the full data set, thus testing their generalization capabilities. Initial experiments were conducted using single runs, due to long function evaluation times, and subsequently, promising candidates were tested more rigorously. It was found that the standard PSOs were not competitive on this task, whereas random dynamic neighborhood PSOs generated the best NNs (best generalization accuracy: 96.5%) and some non-random static neighborhoods also performed well.

This paper demonstrates that there is potential for random dynamic neighborhood PSOs as a tool for finding high-quality solutions for a real-world problem. In view of its use of graph parameters, the method can adapt to other problems that may respond better to different neighborhoods.

2. STRONG GROUND MOTION

When an earthquake occurs, part of the energy released takes the form of elastic waves (called *seismic* waves) that propagate through the Earth. There are two types of seismic waves: *body* and *surface*, and two types of body waves: *primary* (P) waves which involve compression and rarefaction of Earth materials, and *secondary* (S) waves which are essentially rotational waves. P waves travel faster than S, and surface waves are the slowest of the three. Instruments recording these waves produce plots that sometimes clearly show the P and S waves, which are hallmarks of an earthquake. However, such instruments often record waves originating from other sources. Seismic waves can be caused by natural or man-made events. Natural events include phenomenon such as tectonic earthquakes, volcanic tremors, rock falls, collapses of karst cavities and storm microseisms. Man-made events include explosions and vibrations from controlled sources, reservoir-induced earthquakes, mining-induced rock bursts, collapses, and cultural noise, for example traffic and industrial practices [1].

Seismic waves are recorded by two basic types of sensor: *inertial seismometers* which measure ground motion relative to an inertial reference, such as a suspended mass, and *strainmeters* which measure the motion of one point on the ground relative to another. An inertial seismometer is more sensitive to an earthquake signal than a strainmeter [1]. A *short-period* inertial seismometer works on the principle that ground motion produces an inertial force that deflects the reference from its equilibrium position and this displacement can be converted to an electrical signal. The reference is returned to its equilibrium position by a mechanical or a electromagnetic restoring force. A *long period* seismometer is based on the force-balance principle: The inertial mass does not move considerably from its equilibrium position, rather it is balanced as far as possible by an electrically generated force. A *force-balance* accelerometer works on the same principle with minor adjustments. At specific frequencies, the reference moves with the ground through the generation of a feedback force proportional to the ground acceleration.

Strong ground motion is defined as ground accelerations in the range of 1-2 g (where g is the gravitational acceleration of $9.8m/s^s$) [3]. The Kinemetrics Inc. (California USA) K2 strong motion instrument, shown in Figure 1 contains an internal triaxial force-balance accelerometer and a built-in GPS timing system. This is the instrument that was used to collect the data used in this paper. The Seismic Research Unit (SRU) at the University of the West Indies is dedicated to, among other things, the monitoring of strong motion activity in Trinidad and Tobago. This is done using K2 accelerometers, each configured as a four-channel digital recorder that continuously monitors seismic signals. When a signal surpasses a threshold duration magnitude of 2 [8], the signal data (called an *event*) is recorded.

Technicians and researchers subsequently classify and analyze events. This paper focuses on the classification of an event as being caused by a bona fide earthquake or not. The existing method for classification is based on visual inspection and is performed as follows: Obvious non-earthquakes are immediately discarded. Other events are first checked for clear P and S phases, which indicate an earthquake origin. If the P wave is inconspicuous but there is a strong S wave component, the event is flagged for future analysis. If an event possesses ambiguous P and S waves, the date and time the event occurred is matched against weak motion seismometer recordings obtained from other stations in the seismic network, in the vicinity in which the earthquake occurred. If no match is found then the data file is discarded.

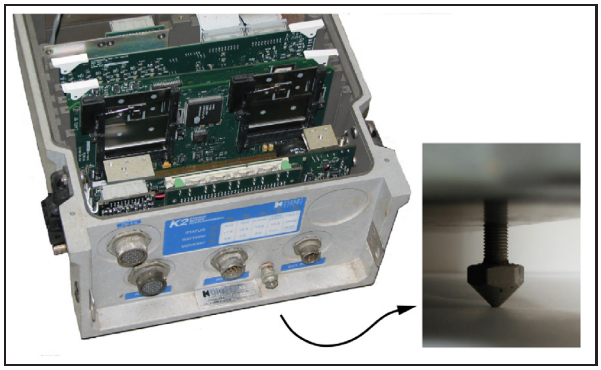


Figure 1: The K2 Strong Motion Digital Recorder

3. PARTICLE SWARM OPTIMIZATION

The particle swarm optimization algorithm is inspired by observations of social interaction [5, 6]. A PSO operates on a population of *particles*, evolving them over a number of iterations with the goal of finding a solution to an optimization function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. Each particle, $P = (x, v, p_b, f_x, f_{p_b})$, carries several pieces of information: a position vector $x \in \mathbb{R}^n$, a velocity vector $v \in \mathbb{R}^n$, a previous best position $p_b \in \mathbb{R}^n$, a fitness value $f_x = f(x)$ and a previous best fitness value $f_{p_b} = f(p_b)$.

Particles in a population, $POP = \{P_0, \ldots, P_{m-1}\}$, are interconnected independent of their positions in the search space. Each particle P has a *neighborhood* $N(P) \subseteq POP$ and each element of $N(P)$ is called a *neighbor* of P. The set $N = \{N(P_0), \ldots, N(P_{m-1})\}$ of neighborhoods of all particles can be represented by a **neighborhood graph** (or **neighborhood structure**) defined by $G_N = (V, E)$ where $V = \{0, 1, \ldots, m-1\}$ and $E = \{(i, j)|P_j \in N(P_i)\}$.

The operation of a PSO is given in Figure 2 and the subroutine `find_neighborhood_best()` is given in Figure 3. K_1 and K_2 are called *individual* and *social* constants.

One very important aspect of the algorithm is the *velocity update equation*: $v = v+\phi_1(p_b-x)+\phi_2(p_*-x)$ Modified versions of this basic equation are common. Clerc and Kennedy [2] introduced the *constriction factor* χ making the equation $v = \chi(v + \phi_1(p_b - x) + \phi_2(p_* - x))$. Typical values used in experimental work are χ=0.729 and $\phi_1, \phi_2 \sim U(0, 2.05)$. Mendes altered the equation so that the particle is influenced by all of its neighbors, not just the best one [10, 11, 9]. This scheme, called the *Fully Informed Particle Swarm* (FIPS) operates as follows. For a particle P, letting

$$\phi_k \sim U\left(0, \frac{\phi_{max}}{|N(P)|}\right) \forall k \in N(P)$$

$$P_{fips} = \frac{\sum_{k \in N} W(k)\phi_k k.p_b}{\sum_{k \in N} W(k)\phi_k}$$

the equation becomes $v = \chi\,(v + \phi(P_{fips} - x))$. ϕ_{max} is usually set to 4.1 and χ to 0.729. $W(k)$ is a weighting function that scales the contributions of the neighbors. It may be desired to weight neighbors' contributions in proportion

```
set t=0
initialize_population(m,n,x_max)
initialize_neighborhoods(m)

while ( NOT termination-condition ) {
  for (i=1; i ≤ m; i++){
    p_* = find_neighborhood_best(N(i))
    for (d=1; d ≤ n; d++){
      φ_1 = K_1 * random(0,1)
      φ_2 = K_2 * random(0,1)

      for particle P_i:
        v_d = v_d + φ_1(p_bd − x_d) + φ_2(p_*d − x_d)
      enforce_maximum_velocity(P_i.v_d, v_max)

      P_i.x_d = P_i.x_d + P_i.v_d
    }
    P_i.f_x = f(P_i.x)
    if (P_i.f_x < P_i.f_pb){
      P_i.p_b = P_i.x
      P_i.f_pb = P_i.f_x
    }
  }
}
```

Figure 2: Pseudocode for a basic PSO algorithm

```
find_neighborhood_best(N){

  idx = α ∈ N, where α is arbitrary.
  for each j ∈ (N − {α})
    if (P_j.f_pb < P_idx.f_pb)
      idx = j
  return idx
}
```

Figure 3: Finding the nbr'hood best of a particle.

to their fitnesses, or their distances from the current particle. Alternatively, with $W(k)$=1, all neighbors contribute equally to the velocity update equation; in this case, P_{fips} is given by:

$$P_{fips} = \frac{\sum_{k \in N} \phi_k k.p_b}{\phi} \quad \text{where,} \quad \phi = \sum_{k \in N} \phi_k$$

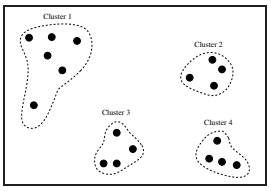
In [10], it was concluded, counter-intuitively, that weighting neighbors' contributions based on fitness does not help the algorithm. In this paper FIPS with equal weighting is used.

3.1 Time Varying Neighborhoods in PSO

In some PSO research, a particle's neighborhood changes with time. In some cases the changes are dependent on the state of other particles, and in other cases it is independent. The goal of changing a particle's neighborhood with time is to manipulate the partners with which it exchanges information. Doing this can affect the overall behavior of the PSO. This subsection gives a brief review of some such work, finishing with more recent approaches that are used in this paper.

3.1.1 Cluster Neighborhoods

Kennedy [4] proposed using *clusters* as an alternative to using an individual's previous best p_b and its neighborhood previous best p_*. A cluster is a set of individuals that are located nearby in the search space (example: Figure 4). Clusters are re-determined at each iteration.



	Replace p_b by	Replace p_* by
1.	p_b	p_*
2.	clus(p_b)	p_*
3.	p_b	clus(p_*)
4.	clus(p_b)	clus(p_*)

Figure 4: An Illustration of Clustering and Experimental Settings

Experiments were performed with the conditions shown in the right of Figure 4, where clus(x) represents the center of the cluster to which x belongs. It was found that conditions involving clus(p_b) performed well, in particular condition #2 in the table.

3.1.2 Gradually adding neighbors

Suganthan [16] suggested gradually changing the focus of the PSO from exploration to exploitation by using a neighborhood operator. A particle P's neighborhood starts empty and near the end of the algorithm, consists of the entire population. $N(P)$ is determined as a function of the current iteration t as follows. A threshold fraction $f = (3.0t + 0.6t_{max})/t_{max}$ is computed, where t_{max} is iteration limit. If f <0.9, then the $N(P)$ is set to the entire population, otherwise, it is taken to consist of all particles whose distance d from P satisfies $d/d_{max} < f$, were d_{max} be the maximum distance between P and any other particle.

3.1.3 Large Ring to Small Global

Richards and Ventura [14] proposed a version of the PSO that starts with a relatively large ring population, and over time removes some of the worst particles and adds additional connections between the remaining particles. A maximum number of function evaluations are allocated to a PSO run, some are reserved for use with the initial ring structure consisting of $size_i$ nodes; this is the *first phase*. Some are reserved for use with the fully connected structure on $size_f$ nodes; this is the *last phase*. The remainder is allocated to a *dynamic phase*, in which after each iteration of the PSO, the population size is reduced and increasingly dense L-Best-k structures are used. If f is the fraction of evaluations used so far in the dynamic phase, then the population size is reduced to $size = (size_f - size_i)(1 - f)^2 + size_f$ by deleting the worst particles. The connection density is increased to $k = 1 + f(size - 1)$ and the new population structure is set to L-Best-k, except that $P \in N(P)$.

3.2 Random Dynamic Neighborhoods

Mohais et al [13, 12] used random neighborhoods in PSO, together with dynamism operators. This is the method that will be used for the application problem in this paper.

Their method for creating a random graph uses two parameters, the number of nodes (n) and the uniform out-degree (k). For node n_i, $N(n_i)$, is randomly chosen so that $N(n_i) = \{nb_1, nb_2, \ldots, nb_k\}$ where $nb_{j_1} \neq nb_{j_2}$ for all distinct $1 \leq j_1, j_2 \leq n$. A directed edge is added from the n_i to each neighbor nb_j and this is taken to mean that n_i has nb_j in its neighborhood, but not vice versa.

Two methods of dynamism called *Random Edge Migration* and *Total Re-Structuring* are given in [13, 12]. These operators works on a neighborhood graph by re-arranging edges, either very little or a lot, while the number of nodes remains fixed, i.e. there is no change in the number of particles in the swarm. The operators are probabilistically applied at the end of each iteration, according to a parameter p_{dyn} called the *dynamism application probability*. p_{dyn} controls how frequently dynamism is used. If p_{dyn}=1.0, then it is assured that at the end of each iteration of the PSO, dynamism would be applied. If p_{dyn}=0.1, dynamism would only be applied about 10% of the time.

3.2.1 Random Edge Migration

Random edge migration removes a randomly selected edge (n_i, n_j) and inserts a randomly selected replacement edge (n_k, n_j). This is interpreted as removing n_j from the $N(n_i)$ and placing it into $N(n_k)$. It is ensured that n_i is such that $|N(n_i)| > 1$ so that upon removing the neighbor n_j, n_i does not become isolated. It is also ensured that $|N(n_k)| \neq n$, i.e. that n_k does not already have a full neighborhood. As edge migrations are performed, the uniform out-degree k characteristic is changed, the distribution of out-degrees becomes varied. Edge migration is illustrated in Figure 5. The edge $(1, 2)$ is randomly selected for removal, and is replaced, again randomly, by $(3, 2)$.

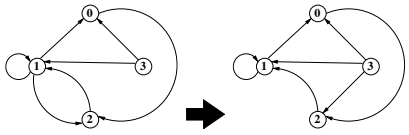


Figure 5: An illustration of random edge migration.

3.2.2 Total Re-Structuring

Total re-structuring is the complete re-positioning of the edges in the neighborhood graph. It is effectively a re-initialization of the random graph based on the parameters n and k. This operation results in a drastically new configuration in which it is unlikely that any of the previous neighborhoods still exist. It is expected that this operation can quickly halt a decline towards premature convergence. An illustration of total re-structuring is shown in Figure 6 using a random graph with $n = 7$ and $k = 2$. After re-structuring, n and k remain the same, but the edges are quite different.

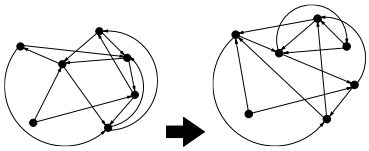


Figure 6: An illustration of total re-structuring.

4. THE PSO/NN STRONG-MOTION APP.

4.1 The Data Set

Once a K2 recorder is triggered, it begins recording a time series of amplitude measurements. The recording is stopped when the motion subsides. Events last for different amounts of time and so the time series vary in length. Figure 7 shows amplitude time series recordings for a short and a long earthquakes. Figure 8 shows two examples of events whose recordings are considered obvious non-earthquakes. Sometimes a non-earthquake event can be ambiguous to classify at first glance, such as in Figure 9.

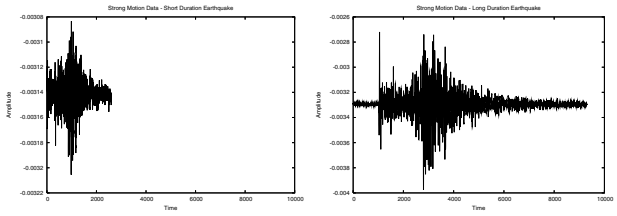


Figure 7: Strong Motion Data for Short and Long Duration Earthquakes

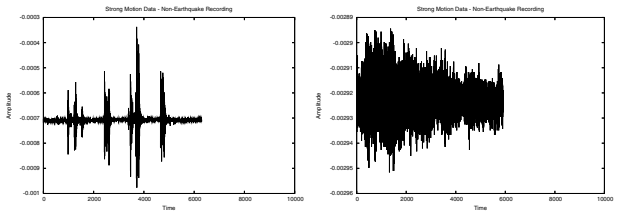


Figure 8: Strong Motion Data for Clear Non-Earthquakes

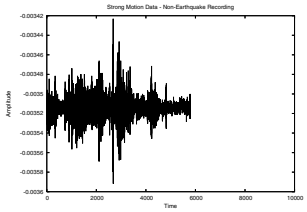


Figure 9: Strong Motion Data for an Ambiguous Non-Earthquake

The SRU maintains recording stations at five locations throughout Trinidad. These locations are shown in the left of Figure 10. The stations regularly capture data from events originating on the island of Trinidad, in the waters off of Trinidad, as well as from much farther away such as in the regions of Barbados, Grenada, St. Vincent and the Grenadines, St. Lucia and Dominica.

The SRU provided a data set of all events recorded at the Brigand Hill station during the year 2004. This data set consisted of 1503 events, of which 113 were actual earthquakes and 1390 were due to other sources. The right of Figure 10 shows a map of the southermost part of the Caribbean with a representative sample of the earthquake events from the data set marked by cross-hairs.

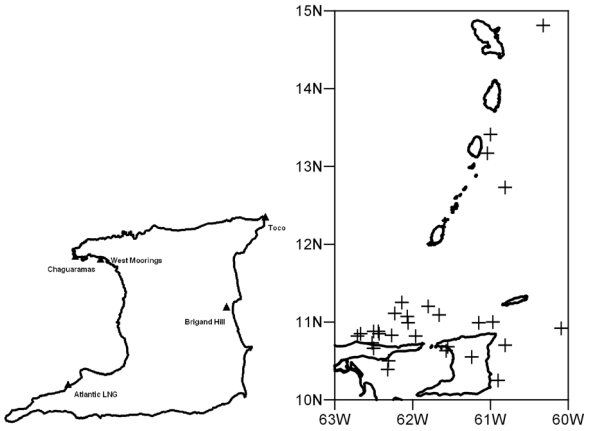


Figure 10: Left: Locations of the Strong Motion Recording Stations in Trinidad. Right: Locations of the Events from the 2004 Data Set.

For the purposes of training and generalization testing, the data set of 1503 events was randomly split into roughly equal halves:

	Set 1	Set 2
# of Earthquakes	56	57
# of Non-earthquakes	695	695

Set 1 was used as a training set for the PSO that evolves neural networks to classify the events. After a neural network was evolved, it was judged by its performance on the entire data set (Sets 1 and 2 together).

4.1.1 Normalization of the Data

Data collected by the strong motion instrument could vary widely in terms of the magnitudes of the accelerations. Thus, two different data files may both represent earthquakes, but the amplitudes of their data points can be of very different magnitudes. This can happen for example, if one of the earthquakes occurred relatively near to the recording instrument and so produced relatively large amplitudes, whereas the other occurred very far away and thus produced relatively small amplitudes. In order to simplify the type of neural network needed, each data file was first normalized so that the mean of all of its points was 0, and the standard deviation 1. This was accomplished using the transformation formula given in equation (1) where in the original data set, μ is the mean, σ is the standard deviation, and x is a data point; x' is the normalized data point.

$$x' = \frac{(x - \mu)}{\sigma} \tag{1}$$

4.1.2 Standardization of Data Length

The events from the data set varied greatly in length, ranging from 2000 to 14000. The distributions of event sizes for the entire data set, as well as for earthquake events and non-earthquake events separately are given in Figure 11 (the scale of the plot for the earthquake events is much smaller than that of the others). Based on these distributions, heuristics could be used to eliminate some data from consideration outright, but this is not done here; effort is spent exclusively developing a neural network capable of making the correct classification based on the input data without any kind of analysis of that data.

Again, in order to simplify the structure of the neural network required to classify these data, all data files were standardized to a common length before being loaded into the PSO. This length, called the **input length** was a variable in the experiments performed to obtain the best possible neural network. In all cases considered, the standardized length was smaller than the length of the smallest data file. The standardization was done by using an averaging window to replace a set of adjacent data points by a single averaged point. The algorithm for doing this is given in Figure 12, this algorithm assumes that the new set of data points is smaller than the original set. An example of an event at its original length of and at a shortened length of 500 is shown in Figure 13; the general shape of the data is preserved.

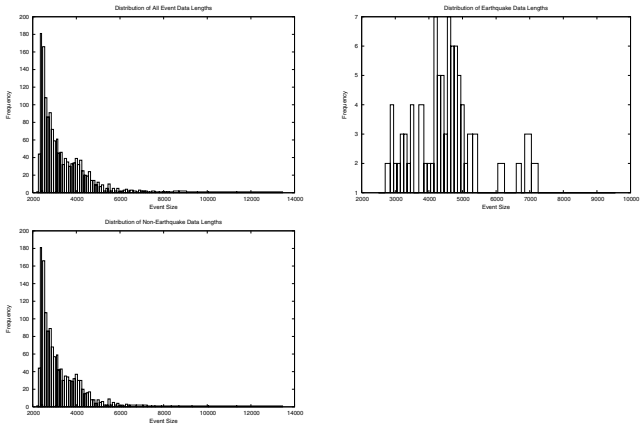


Figure 11: Distribution of earthquake data sizes for entire data set.

```
shorten_data_length(data[], shortened_size){
  averaging_window = data.length/shortened_size;
  for (i = 0; i < shortened_size; i++){
    start = i * averaging_window;
    stop = start + averaging_window;
    sum = data[start];
    for (j = start + 1; j < stop; j++)
      sum += data[j];
    shortened_data[i] = sum / averaging_window;
  }
  return shortened_data;
}
```

Figure 12: Shortening a collection of data points.

4.2 The PSO/NN Setup

4.2.1 Representing a Neural Network as a Particle

Given a fixed network architecture (i.e. number of input, hidden and output nodes), a PSO is set up in which a particle's position represents a possible set of weights for that architecture. Weights for the input to hidden layer are

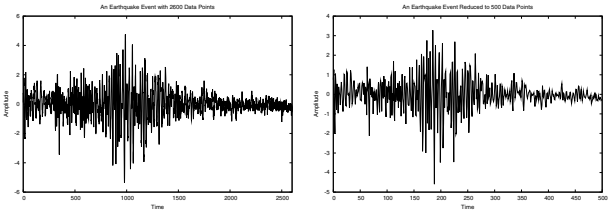


Figure 13: 2600 data points reduced to 500

placed in the first part of the position vector, and are followed by the weights for the hidden to output nodes. This mapping is illustrated in Figure 14.

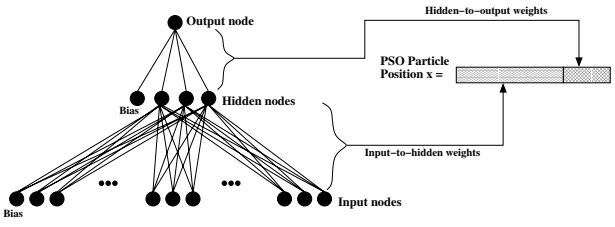


Figure 14: Neural network weights mapped to the position vector of a PSO particle.

4.2.2 *Defining the Optimization Function*

To evaluate a particle, the neural network that it represents is applied to the first half of the partitioned data set (Set 1). A direct approach would be to take the number of correctly classified events as the particle's fitness value; this would be in the interval $[0, 1503]$. However the data set has a severe imbalance between the number of positive (113) and negative (1390) examples, and so adopting this approach would likely cause a rush into a region of the search space in which the positive examples are all misclassified. Instead, it was decided to weight the positive examples so as to compensate for their small number, whence fitness was calculated using:

$$fitness = -1 * (n_{pos} * w_{pos} + n_{neg}) \quad (2)$$

n_{pos} and n_{neg} are the number of correctly classified positive and negative examples respectively and w_{pos} is the compensatory weight assigned to positive examples, $w_{pos} = \frac{\#\text{ negative examples}}{\#\text{ positive examples}} = \frac{695}{56}$. A negative sign is used in the fitness formula make the problem one of minimization. The goal of the PSO is thus to find a set of neural network weights, which when used on an appropriate architecture to classify the training data, yields a 100% correct classification. Such a set of weights would have a fitness value $-1 \cdot (56 \cdot \frac{695}{56} + 695) = -1390$.

5. PSO/NN TRAINING EXPERIMENTS

5.1 Experimental Setup

Experiments were conducted in an attempt to find a neural network that would correctly **generalize** over the entire data set. A FIPS-based PSO with re-structuring and a linearly decreasing dynamism probability, from p_{dyn}=1.0 down to 0.0 over a run, was used. Different neighborhood parameters, i.e. number of particles (m) and uniform out-degree (k), and NN architectures, i.e. number of input nodes (n_i) and hidden nodes (n_h), were tested. The choice of architecture parameters was based on probing experiments. The parameters were:

- $n_i \in \{200, 210, 220, \ldots, 350\}$
- $n_h \in \{15, 16, \ldots, 20\}$
- $m \in \{20, 30\}$
- $k \in \{1, 2, \ldots, 10\}$

For comparison, the static neighborhoods L-Best-1/2/3/4 and von Neumann, were tested with the same parameters as above, except for k which has no meaning in this context. Because of the nature of this real-world application, evaluating the fitness of a particle takes a very long time. Thus, to maintain a feasible experiment, only a *single* run of each configuration was executed. Each run was allowed a maximum of 25000 evaluations. Subsequently, for special cases, multiple-trial experiments were performed to obtain statistically valid comparisons. This approach was considered acceptable since the objective of the experiment was to obtain a NN that could correctly classify, not necessarily to be able to reproduce experimental runs. For each configuration, the best particle at the end of the PSO run was kept and evaluated on both the test set and the entire data set.

5.2 Results

Tables 1 and 2 show the top 10 results (11 shown due to a tie) for the random dynamic PSOs and for the static ones respectively. These tables present two separate sets of results for training and generalization. In each category, n_{pos}, n_{neg} and n_{tot} refer to the number of correctly classified positive (earthquakes), negative (non-earthquakes) and total events (sum of the previous two) respectively. The tables are sorted on n_{tot} for generalization. In each class the best results are highlighted in gray. Training results alone would not have been enough to differentiate the performances of the various configurations and furthermore, the NNs that obtained the best training results did not also generalize best.

The standard PSO neighborhood (L-Best-1) was not in Table 2 and so was clearly out-performed. However the von Neumann neighborhood produced an impressive performance in comparison to the random dynamic neighborhoods. This is in contrast to results on many typical research problems where it is out-performed [12]. Nevertheless, the best random dynamic neighborhood configuration from Table 1 produced a better generalizer than the best from Table 2, and further experiments confirmed this statistically.

Figure 15 shows how generalization performance behaved in relation to k for both levels of n (20 and 30) for the top-4 NN architectures from Table 1. There is a peak in performance in the region of k=4,5. This is further evidenced in Table 3, which shows the k values at which the best NNs were obtained for each architecture. Even though the data represents single runs of each configuration, it is striking that the values of k are mostly 4 or 5. Based on this observation, neighborhoods using k=4,5 were examined to see how generalization performance behaved in response to architecture. This is shown in the contour plots of Figure 16, where darker shades represent better performance. The scattered dark patches suggest that when k=4,5, there is no range of architectures that is particularly better. However, there are more architectures that perform well with k=4, than with 5.

				Testing			Generalization		
n_i	n_h	m	k	n_{pos}	n_{neg}	n_{tot}	n_{pos}	n_{neg}	n_{tot}
340	19	30	4	56	688	**744**	85	1366	**1451**
350	16	30	5	55	690	**745**	86	1362	**1448**
320	18	30	4	55	686	**741**	86	1360	**1446**
320	15	20	4	52	687	**739**	76	1370	**1446**
350	15	30	4	56	688	**744**	88	1357	**1445**
350	20	30	4	54	685	**739**	78	1367	**1445**
290	19	30	5	56	692	**748**	77	1367	**1444**
310	15	30	5	56	690	**746**	81	1361	**1442**
340	17	30	4	56	687	**743**	79	1363	**1442**
320	18	20	4	56	689	**745**	82	1359	**1441**
300	19	30	6	55	690	**745**	78	1363	**1441**

Table 1: Best neural networks obtained using random dynamic neighborhoods.

				Testing			Generalization		
n_i	n_h	m	nbd	n_{pos}	n_{neg}	n_{tot}	n_{pos}	n_{neg}	n_{tot}
350	16	30	von	56	688	**744**	82	1354	**1436**
330	19	30	von	55	691	**746**	79	1347	**1426**
320	15	30	von	55	679	**734**	80	1337	**1417**
320	19	30	von	55	685	**740**	74	1341	**1415**
240	18	30	von	56	685	**741**	78	1336	**1414**
300	20	30	von	56	676	**732**	88	1325	**1413**
290	20	30	von	56	679	**735**	84	1329	**1413**
350	17	20	lbest2	55	679	**734**	88	1324	**1412**
330	18	30	von	56	686	**742**	75	1337	**1412**
290	17	30	von	56	675	**731**	85	1326	**1411**
290	18	30	von	56	676	**732**	81	1329	**1410**

Table 2: Best neural networks obtained using static neighborhoods.

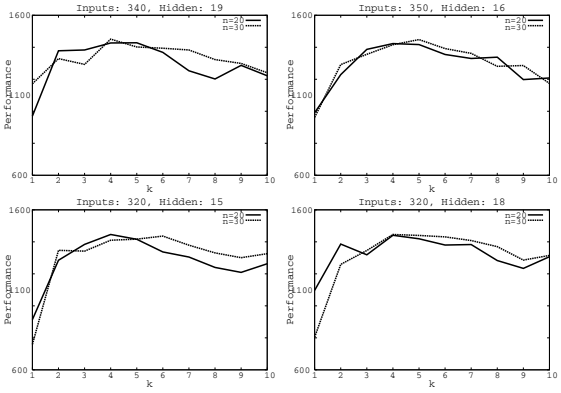


Figure 15: k versus $performance$ plots for the top-four architectures

	n_h					
n_i	15	16	17	18	19	20
200	5	4	5	4	6	7
210	4	4	4	4	4	5
220	4	5	4	4	5	6
230	4	6	4	5	4	6
240	5	4	5	7	4	4
250	4	4	4	5	6	5
260	4	4	5	4	5	4
270	6	5	5	5	6	5
280	4	5	6	5	4	4
290	5	5	4	4	5	4
300	4	5	5	5	6	5
310	5	5	5	4	4	6
320	4	4	5	4	5	6
330	4	5	5	5	5	5
340	4	5	4	6	4	6
350	4	5	4	4	5	4

Table 3: k values at which best NNs were obtained.

5.3 Statistical Comparison of Bests

As a final test, the best configurations of Tables 1 and 2 were put through 100 trial runs so that a statistically sound conclusion could be drawn. The first configuration is a random dynamic FIPS-based PSO using n=30 and k=4 on the architecture n_i=340 and n_h=19. The second configuration is a standard static FIPS-based PSO using a von Neumann graph with n=30 on the architecture n_i=350 and n_h=16.

The results are shown in Table 4 which gives the 95% confidence intervals for the n_{tot} values for generalization. It is clear in both cases that the top results obtained in Tables 1 and 2 happened to be on the high ends of the samples of results. Nevertheless, the best random dynamic PSO outperforms the best standard static one decisively.

6. CONCLUSIONS

None of the many configurations tried was able to produce a neural network capable of generalizing 100% correctly. The best result was an accuracy of 96.5%, obtained by a random dynamic neighborhood PSO. At first glance it

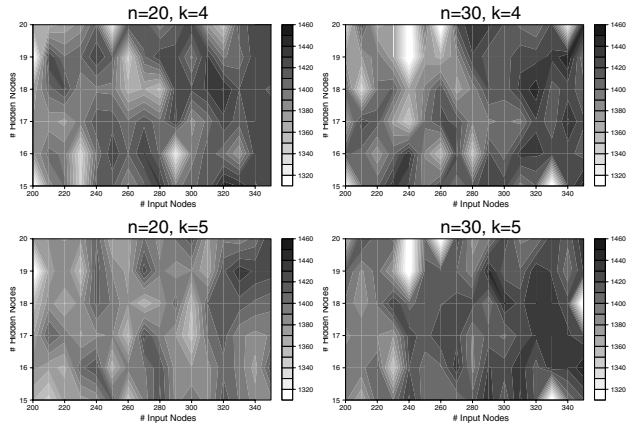


Figure 16: Performance in relation to architecture when k=4,5

Random Dynamic	Standard Static
[1420.910, 1427.370]	[1388.868, 1397.872]

Table 4: 95% Confidence intervals of the best random dynamic configuration versus that of the best standard static configuration.

seemed that for this application, the random dynamic PSO was only marginally better than the standard one using von Neumann. However a more careful examination demonstrated that the difference is significant and much larger than it appeared at first. Many of the top-performing configurations used graphs with out-degree 4 and even the best standard static PSO used the von Neumann topology which also has k=4. It seems plausible that k=4 might be near optimal for this particular problem. These results corroborate the suggestion in [9] that k=4 might be an optimal value for problems that are typically encountered in scientific research and applications.

A classification accuracy of 96.5% is remarkable for a practical application. It remains possible that a perfect classifier could be evolved if a better neural network architecture were chosen. This is a source of future research. Another explanation exists for the 3.5% deficiency - it is that the Seismic Research Unit from which the data set was obtained classifies some of the event using secondary data sources, i.e. data collected from instruments other than the strong motion accelerometers. This external data was not part of the data set used and thus not accessible to the neural networks. It is possible that if given access to this additional data, better accuracy could be attained. Yet other approaches such as *bagging* may improve the performance to 100%.

Due to the very large amounts of time required to evaluate particle fitnesses, the initial experiments were restricted to single runs. This aspect of the study needs to be improved. This work is currently being undertaken.

7. REFERENCES

[1] P. Bormann, editor. *IASPEI New Manual of Seismological Observatory Pratice. Volume I.* GeoForschungsZentrum Potsdam, 2002.

[2] M. Clerc and J. Kennedy. The particle swarm - explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[3] J. Havskov and G. Alguacil. *Instrumentation in Earthquake Seismology*. Springer, Berlin, 2004.

[4] J. Kennedy. Stereotyping: Improving particle swarm performance with cluster analysis. In *Proc. of the IEEE Congress on Evolutionary Computation*, pages 1507–1512, 2000.

[5] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Proc. of IEEE International Conference on Neural Networks*, pages 1942–1948. IEEE Press, 1995.

[6] J. Kennedy, R. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[7] J. Kennedy and R. Mendes. Population structure and particle swarm performance. In *Proc. of the IEEE Congress on Evolutionary Computation*, pages 1671–1676, 2002.

[8] Kinemetrics Inc. *K2 Digital Recorder User Manual - Document 302200 Revision E*. 222 Vista Avenue, Pasadena CA 91107 USA, 1998.

[9] R. Mendes. *Population Toplogies and Their Influence in Particle Swarm Performance*. PhD thesis, Universidade do Minho, Braga, Portugal, February 2004.

[10] R. Mendes, J. Kennedy, and J. Neves. Watch thy neighbor or how the swarm can learn from its environment. In *Proc. of the Swarm Intelligence Symposium*, pages 88–94, 2003.

[11] R. Mendes, J. Kennedy, and J. Neves. The fully informed particle swarm: Simpler, maybe better. *IEEE Transactions on Evolutionary Computation*, 8(3):204–210, 2004.

[12] A. S. Mohais, R. Mendes, C. Ward, and C. Posthoff. Neighborhood re-structuring in particle swarm optimization. In *Lecture Notes in Artificial Intelligence (AI 2005: Advances in Artificial Intelligence)*.

[13] A. S. Mohais, C. Ward, and C. Posthoff. Randomized directed neighborhoods with edge migration in particle swarm optimization. In *Proc. of the IEEE Congress on Evolutionary Computation*, pages 548–555, 2004.

[14] M. Richards and D. Ventura. Dynamic sociometry and population size in particle swarm optimization. *Proc. of the Sixth International Conference on Computational Intelligence and Natural Computing*, pages 1557–1560, 2003.

[15] J. B. Shepherd and W. P. Aspinall. Seismicity and earthquake hazard in trinidad and tobago, west indies. *Earthquake Engineering and Structural Dynamics*, 11:229–250, 1983.

[16] P. N. Suganthan. Particle swarm optimiser with neighbourhood operator. In *Proc. of the IEEE Congress on Evolutionary Computation*, pages 1958–1962. IEEE Press, 1999.

[17] M. D. Trifunac and M. I. Todorovska. Evolution of accelerographs, data processing, strong motion arrays and amplitude and spatial resolution in recording strong earthquake motion. *Soil Dynamics and Earthquake Engineering*, 21(6):537–555, 2001.