Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής

Πρόγραμμα Μεταπτυχιακών Σπουδών

«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Χρήση Εξελικτικών Αλγορίθμων για την εκπαίδευση Τεχνητών Νευρωνικών Δικτύων** **Evolutionary Algorithms for training Neural Networks**
Ονοματεπώνυμο Φοιτητή	**ΝΙΚΟΛΑΟΣ – ΣΠΥΡΙΔΩΝ ΑΝΑΣΤΑΣΙΟΥ**
Πατρώνυμο	**ΜΙΧΑΗΛ**
Αριθμός Μητρώου	**ΜΠΠΛ 13003**
Επιβλέπων	**ΘΕΜΗΣ ΠΑΝΑΓΙΩΤΟΠΟΥΛΟΣ , ΚΑΘΗΓΗΤΗΣ**

Ημερομηνία Παράδοσης : **Νοέμβριος 2015**

Περίληψη

Η παρούσα διπλωματική εργασία εξετάζει εξελικτικές τεχνικές για την εκπαίδευση Τεχνητών Νευρωνικών Δικτύων (ΤΝΔ) . Με τον όρο εξελικτική τεχνική αναφερόμαστε σε μια υποκατηγορία των μετα - ευριστικών τεχνικών . Οι εξελικτικοί αλγόριθμοι περιλαμβάνουν τεχνικές οι οποίες προσπαθούν να προσομοιώσουν συμπεριφορές πληθυσμών που απαντώνται στην φύση , όπως για παράδειγμα την διαδικασία της εξέλιξης των ειδών , την κίνηση των πουλιών από ένα σημείο σε ένα άλλο, την εύρεση τροφής από ένα κοπάδι ψαριών, κα . Ο συμβατικός τρόπος εκπαίδευσης ενός ΤΝΔ γίνεται με την χρήση του αλγορίθμου Οπισθοδιάδοσης του Σφάλματος (BP – Back Propagation Error) . Στην παρούσα εργασία ο αλγόριθμος BP συγκρίνεται με 3 εξελικτικές τεχνικές: τον Γενετικό Αλγόριθμο (ΓΑ) , τον αλγόριθμο PSO και μία υβριδική τους προσέγγιση η οποία ονομάζεται HGAPSO . Για την αποδοτική σύγκριση των 4 αλγορίθμων υλοποιήθηκε λογισμικό , το οποίο εφαρμόζει και τις 4 τεχνικές πάνω σε σύνολα δεδομένων . Τα σύνολα δεδομένων τα οποία χρησιμοποιήθηκαν ανακτήθηκαν από το UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.html) . Τα δεδομένα εκπαίδευσης μετασχηματίστηκαν κατάλληλα , σε ισοδύναμη μορφή, έτσι ώστε να είναι δυνατός ο χειρισμός τους από το λογισμικό .

Abstract

The current thesis examines evolutionary techniques for the training of Artificial Neural Networks (ANN) . The term evolutionary technique refers to a subset of post-heuristic techniques . Evolutionary algorithms include techniques which try to simulate behaviors of populations found in nature , such as the process of evolution of species , the movement of birds from one place to another , finding food in fish-swarm , and more . The conventional way of training an ANN is using the back-propagation algorithm (BP - Back Propagation) . In this work the algorithm BP is compared with 3 evolutionary techniques : the Genetic Algorithm (GA) , the PSO algorithm and a hybrid approach them called HGAPSO . For efficient comparison of these four algorithms we implemented a software which applies the four techniques on data sets . The data sets that were used were recovered from the UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.html) . The training data properly transformed, in an equivalent form so as to be used by the software .

ΠΕΡΙΕΧΟΜΕΝΑ

ΚΕΦΑΛΑΙΟ 1: ΕΙΣΑΓΩΓΗ

Η παρούσα διπλωματική εργασία εξετάζει εξελικτικές τεχνικές για την εκπαίδευση Τεχνητών Νευρωνικών Δικτύων (ΤΝΔ) . Με τον όρο εξελικτική τεχνική αναφερόμαστε σε μια υποκατηγορία των μετα - ευριστικών τεχνικών . Οι εξελικτικοί αλγόριθμοι περιλαμβάνουν τεχνικές οι οποίες προσπαθούν να προσομοιώσουν συμπεριφορές πληθυσμών που απαντώνται στην φύση , όπως για παράδειγμα την διαδικασία της εξέλιξης των ειδών , την κίνηση των πουλιών από ένα σημείο σε ένα άλλο , την εύρεση τροφής από ένα κοπάδι ψαριών , κα .

Ο συμβατικός τρόπος εκπαίδευσης ενός ΤΝΔ γίνεται με την χρήση του αλγορίθμου Οπισθοδιάδοσης του Σφάλματος (BP – Back Propagation Error) . Στην παρούσα εργασία ο αλγόριθμος BP συγκρίνεται με 3 εξελικτικές τεχνικές: τον Γενετικό Αλγόριθμο (ΓΑ) , τον αλγόριθμο PSO και μία υβριδική τους προσέγγιση η οποία ονομάζεται HGAPSO .

Για την αποδοτική σύγκριση των 4 αλγορίθμων υλοποιήθηκε λογισμικό , το οποίο εφαρμόζει και τις 4 τεχνικές πάνω σε σύνολα δεδομένων . Τα σύνολα δεδομένων τα οποία χρησιμοποιήθηκαν ανακτήθηκαν από το UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.html) . Τα δεδομένα εκπαίδευσης μετασχηματίστηκαν κατάλληλα , σε ισοδύναμη μορφή , έτσι ώστε να είναι δυνατός ο χειρισμός τους από το λογισμικό .

Η δομή της διπλωματικής εργασίας είναι η ακόλουθη :

- **Κεφάλαιο 1 : Εισαγωγή .** Στο κεφάλαιο αυτό δίνεται μία σύντομη περιγραφή της διπλωματικής εργασίας .
- **Κεφάλαιο 2 : Εξελικτικοί Αλγόριθμοι .** Στο κεφάλαιο αυτό ορίζεται η έννοια του εξελικτικού αλγορίθμου και δίνεται ιδιαίτερη βαρύτητα στους δύο δημοφιλέστερους εξελικτικούς αλγορίθμους : στον Γενετικό Αλγόριθμο και στον αλγόριθμο PSO .
- **Κεφάλαιο 3 : Τεχνητά Νευρωνικά Δίκτυα και Μέθοδοι Εκπαίδευσης .** Στο κεφάλαιο αυτό περιγράφεται η έννοια του ΤΝΔ και της εκπαίδευσής του . Πιο συγκεκριμένα , περιγράφεται ο συνηθέστερος αλγόριθμος , αυτός της Οπισθοδιάδοσης του Λάθους .
- **Κεφάλαιο 4 : Εξελικτικές τεχνικές για την εκπαίδευση Τεχνητών Νευρωνικών Δικτύων .** Στο κεφάλαιο αυτό περιγράφονται παραλλαγές του Γενετικού Αλγορίθμου και του αλγορίθμου PSO για την εκπαίδευση ΤΝΔ .
- **Κεφάλαιο 5 : Μια υβριδική εξελικτική τεχνική για την εκπαίδευση Τεχνητών Νευρωνικών Δικτύων .** Στο κεφάλαιο αυτό περιγράφεται η δομή του προτεινόμενου αλγορίθμου HGAPSO .
- **Κεφάλαιο 6: Μεθοδολογία σύγκρισης των αλγορίθμων εκπαίδευσης ΤΝΔ .** Στο κεφάλαιο αυτό περιγράφεται η μεθοδολογία η οποία θα ακολουθηθεί για την σύγκριση των 4 μεθόδων εκπαίδευσης ΤΝΔ .
- **Κεφάλαιο 7 : Λογισμικό εκτέλεσης πειραμάτων .** Στο κεφάλαιο αυτό παρουσιάζεται η δομή του λογισμικού εκτέλεσης πειραμάτων, καθώς κι ένα εγχειρίδιο χρήσης .
- **Κεφάλαιο 8 : Πειραματικά αποτελέσματα .** Στο κεφάλαιο αυτό παρουσιάζονται τα αποτελέσματα των πειραμάτων , τα οποία εκτελέστηκαν με την χρήση του λογισμικού .
- **Κεφάλαιο 9 : Συμπεράσματα – Επεκτάσεις .** Στο κεφάλαιο αυτό παρουσιάζονται τα συμπεράσματα τα οποία εξήχθησαν από τα πειραματικά αποτελέσματα καθώς και πιθανές μελλοντικές επεκτάσεις της παρούσας εργασίας .

ΚΕΦΑΛΑΙΟ 2: ΕΞΕΛΙΚΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ

2.1 Η έννοια του εξελικτικού αλγορίθμου

Με τον όρο εξελικτικοί αλγόριθμοι αναφερόμαστε σε μια υποκατηγορία των μετα - ευριστικών αλγορίθμων . Οι εξελικτικοί αλγόριθμοι περιλαμβάνουν τεχνικές οι οποίες προσπαθούν να προσομοιώσουν συμπεριφορές πληθυσμών που απαντώνται στην φύση , όπως για παράδειγμα την διαδικασία της εξέλιξης των ειδών , την κίνηση των πουλιών από ένα σημείο σε ένα άλλο , την εύρεση τροφής από ένα κοπάδι ψαριών , κα .

Με τον όρο μετα - ευριστικός αλγόριθμος αναφερόμαστε σε μία κατηγορία επαναληπτικών αλγορίθμων οι οποίοι είναι κατάλληλοι για την αναζήτηση λύσεων μέσα σε έναν πολύ μεγάλο χώρο αναζήτησης (ο οποίος επίσης ονομάζεται πεδίο του προβλήματος) . Στην περίπτωση όπου ο χώρος αναζήτησης είναι αρκετά μεγάλος σε μέγεθος, τότε η εξαντλητική αναζήτησή του είναι κάτι το οποίο είναι υπολογιστικά και χρονικά ασύμφορο . Έτσι λοιπόν οι μετα - ευρετικοί αλγόριθμοι πραγματοποιούν μία « έξυπνη » αναζήτηση στον χώρο αναζήτησης με αποτέλεσμα να μην ψάχνουν όλες τις πιθανές λύσεις αλλά ένα υποσύνολο αυτών . Η λύση η οποία επιστρέφεται δεν είναι συνήθως η βέλτιστη αλλά μία λύση αρκετά κοντά σε αυτή . Σε γενικές γραμμές υπάρχει η εγγύηση ότι στην μέση περίπτωση η επιστρεφόμενη λύση θα είναι κοντά στην βέλτιστη μέσα στα όρια μιας μικρής , προκαθορισμένης απόκλισης . Η χρήση των εξελικτικών αλγορίθμων είναι προτιμότερη έναντι της εξαντλητικής αναζήτησης σε προβλήματα όπου δεν υπάρχει απαίτηση για μεγάλη ακρίβεια στην εξεύρεση μιας λύσης (π.χ. εύρεσης του σημείου εγκατάστασης μιας τηλεφωνικής κεραίας μέσα σε ένα νόμο) .

Όλοι οι εξελικτικοί αλγόριθμοι έχουν σε γενικές γραμμές τον ίδιο βασικό τρόπο λειτουργίας και αυτός δεν είναι άλλος από την εξέλιξη ενός πληθυσμού πιθανών λύσεων . Ας θεωρήσουμε έναν οποιοδήποτε χώρο αναζήτησης PS του οποίου το μέγεθος είναι αρκετά μεγάλο ή ακόμη και μη φραγμένο . Κάθε εξελικτικός αλγόριθμος επιλέγει αρχικά ένα υποσύνολο του χώρου αναζήτησης το οποίο ονομάζεται πληθυσμός και συμβολίζεται με P . Το μέγεθος του πληθυσμού θα πρέπει να είναι αρκετά μικρό σε σχέση με το μέγεθος του χώρου αναζήτησης γιατί σε διαφορετική περίπτωση ο εξελικτικός αλγόριθμος θα εκφυλιστεί σε εξαντλητική αναζήτηση . Η επιλογή του αρχικού πληθυσμού P μπορείτε να γίνει είτε με τυχαίο τρόπο είτε με την εφαρμογή κάποιας ευριστικής μεθόδου , έτσι ώστε οι αρχικές πιθανές λύσεις που θα επιλεχθούν να μην εντελώς τυχαίες αλλά αντίθετα όσο το δυνατόν καλύτερες . Έχει επικρατήσει να ονομάζουμε τις λύσεις του πληθυσμού ως άτομα . Μετά την επιλογή του αρχικού πληθυσμού σε κάθε εξελικτικό αλγόριθμο εφαρμόζονται επαναληπτικά τουλάχιστον μία από τις παρακάτω ενέργειες :

- Αξιολόγηση
- Επιλογή
- Διασταύρωση
- Μετάλλαξη

Με τον όρο αξιολόγηση αναφερόμαστε στην ποσοτικοποίηση της ποιότητας του κάθε ατόμου του πληθυσμού . Για τον σκοπό αυτό συνήθως χρησιμοποιείται μία συνάρτηση , η οποία ονομάζεται συνάρτηση αξιολόγησης ή συνάρτηση εύρεσης καταλληλότητας f. Η συνάρτηση αυτή λαμβάνει ως είσοδο ένα άτομο του πληθυσμού και επιστρέφει μία θετική πραγματική τιμή . Η συνάρτηση θα πρέπει να είναι γνησίως αύξουσα σε σχέση με την ποιότητα του ατόμου. Αυτό πρακτικά σημαίνει ότι για δύο άτομα του πληθυσμού A και B, αν το A είναι περισσότερο ποιοτικό από το άτομο B τότε θα πρέπει $f(A) > f(B)$. Η συνάρτηση αξιολόγησης θα πρέπει επίσης να μην έχει μεγάλη υπολογιστική πολυπλοκότητα , αφού η αξιολόγηση του πληθυσμού λαμβάνει χώρα σε κάθε επανάληψη του εξελικτικού αλγορίθμου .

Η διαδικασία της επιλογής λαμβάνει χώρα πάντοτε μετά την διαδικασία της αξιολόγησης . Κατά την επιλογή επιλέγονται κατά βάση τα ποιοτικότερα άτομα του πληθυσμού , δηλαδή αυτά με τις μεγαλύτερες τιμές αξιολόγησης . Ο τρόπος επιλογής θα πρέπει να είναι τέτοιος έτσι ώστε να μην διασφαλίζει την ποικιλομορφία του πληθυσμού . Για παράδειγμα σε ένα κοπάδι ψαριών το οποίο αναζητά τροφή μέσα στην θάλασσα είναι λογικό κάποια ψάρια να είναι καλύτερα από κάποια άλλα. Από την στιγμή που ο εξελικτικός αλγόριθμος προσομοιώνει μία διαδικασία που

απαντάται στην φύση , θα πρέπει να διασφαλίζει την όσο δυνατόν καλύτερη προσομοίωση αυτής . Συνεπώς κατά την επιλογή ναι μεν θα πρέπει να επιλέγονται τα καλύτερα άτομα του πληθυσμού αλλά θα πρέπει επίσης να δίνονται ευκαιρίες και στα λιγότερο κατάλληλα . Τα άτομα του πληθυσμού τα οποία επιλέγονται κατά την διαδικασία της επιλογής συνεχίζουν έπειτα με τις διαδικασίες της διασταύρωσης και της μετάλλαξης .

Η διαδικασία της διασταύρωσης εκτελείται πάνω στα άτομα του πληθυσμού τα οποία προέκυψαν από την διαδικασία της επιλογής . Η διασταύρωση είναι πρακτικά εκείνη η οποία κάνει τον εξελικτικό αλγόριθμο να ανακαλύψει νέες λύσεις μέσα στον χώρο αναζήτησης . Η διασταύρωση χρησιμοποιεί 2 ή περισσότερα από τα επιλεχθέντα άτομα και συνδυάζει τα χαρακτηριστικά τους έτσι ώστε να δημιουργήσει νέα άτομα τα οποία αντιστοιχούν σε πιθανές λύσεις του χώρου αναζήτησης οι οποίες δεν έχουν ακόμη εξεταστεί . Τα άτομα τα οποία χρησιμοποιούνται για την δημιουργία των νέων ατόμων ονομάζονται γονείς ενώ τα νέα άτομα ονομάζονται απόγονοι . Η λογική της διασταύρωσης είναι η εξής : αν οι γονείς είναι καλοί (δηλαδή έχουν καλή τιμή αξιολόγησης) τότε κατά πάσα πιθανότητα και οι απόγονοί τους θα είναι καλοί .

Μετά την διαδικασία της διασταύρωσης ακολουθεί η διαδικασία της μετάλλαξης . Η διαδικασία αυτή επιλέγει τυχαία κάποια άτομα του πληθυσμού και μεταβάλλει ένα ή περισσότερα από τα χαρακτηριστικά τους . Με τον τρόπο αυτό μία ήδη υπάρχουσα λύση μετατρέπεται σε μία άλλη , ίσως μάλιστα και αρκετά διαφορετική . Η νέα λύση μπορεί να είναι καλύτερη ή χειρότερη της λύσης που μεταλλάχθηκε . Ο λόγος ύπαρξης της διαδικασίας της μετάλλαξης είναι διπλός . Πρώτον , με τον τρόπο αυτό ο αλγόριθμος εξετάζει νέες περιοχές του χώρου αναζήτησης , οι οποίες ίσως έχουν ποιοτικές λύσεις και για την ώρα ο αλγόριθμος δεν τις έχει επισκεφθεί . Δεύτερον , ένα αρκετά συχνό πρόβλημα των εξελικτικών αλγορίθμων είναι ότι είναι δυνατόν ο πληθυσμός να παγιδευτεί σε ένα τοπικό βέλτιστο του χώρου αναζήτησης . Με τον τρόπο αυτό η μετάλλαξη λειτουργεί σε δικλίδα ασφαλείας σε τέτοιες περιπτώσεις , αφού ακόμη και στην περίπτωση που όλα τα άτομα του πληθυσμού βρίσκονται κοντά σε αυτό το τοπικό βέλτιστο , με την μετάλλαξη κάποιο ή κάποια από αυτά μετακινούνται σε νέες περιοχές του χώρου αναζήτησης . Φυσικά η πιθανότητα μετάλλαξης κάθε ατόμου του πληθυσμού θα πρέπει να είναι μικρή γιατί σε διαφορετική περίπτωση ο αλγόριθμος εκφυλίζεται σε τυχαία αναζήτηση .

Οι 4 παραπάνω λειτουργίες (αξιολόγηση , επιλογή , διασταύρωση , μετάλλαξη) εκτελούνται επαναληπτικά σε κάθε εξελικτικό αλγόριθμο μέχρι να ικανοποιηθεί κάποιο κριτήριο τερματισμού της όλης διαδικασίας . Συνήθη κριτήρια τερματισμού είναι τα παρακάτω :

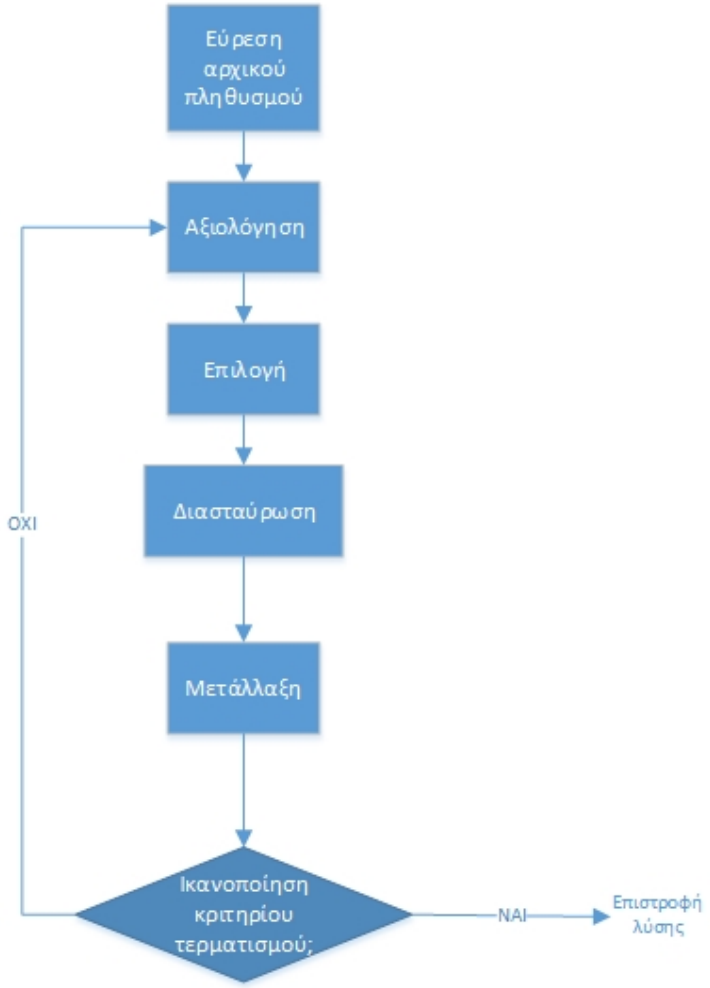
1. Ο αλγόριθμος ολοκλήρωσε έναν συγκεκριμένο αριθμό επαναλήψεων .
2. Το καλύτερο άτομο του πληθυσμού έχει ως τιμή αξιολόγησης μία αρκετά μεγάλη τιμή .
3. Η μέση τιμή αξιολόγησης του πληθυσμού είναι αρκετά μεγάλη .

Μία αναπαράσταση της λογικής των εξελικτικών αλγορίθμων σε ψευδοκώδικα παρουσιάζεται παρακάτω .

```
// Είσοδος
   • το πεδίο ορισμού του προβλήματος PS ( χώρος αναζήτησης )
   • μία συνάρτηση ελέγχου της ποιότητας κάθε υποψήφιας λύσης f ( συνάρτηση
     αξιολόγησης )
// Επεξεργασία
   1. Επέλεξε με κάποια διαδικασία (τυχαία ή μη) τον αρχικό πληθυσμό
      P, ο οποίος αποτελεί γνήσιο υποσύνολο του χώρου αναζήτησης PS
      (P ⊂ PS).
   2. Όσο το κριτήριο τερματισμού της διαδικασίας δεν έχει
      ικανοποιηθεί επανάλαβε τις παρακάτω ενέργειες:
         a. Αξιολόγησε κάθε άτομο του πληθυσμού x ∈ P κάνοντας χρήση
            της συνάρτησης αξιολόγησης f(x).
         b. Επέλεξε τα καλύτερα άτομα του πληθυσμού με βάση την τιμή
            της βαθμολογίας που αυτά έλαβαν από την f(x).
         c. Εκτέλεσε διασταύρωση στα άτομα του παραπάνω βήματος,
            χρησιμοποιώντας κάποια καλά ορισμένη μέθοδο.
```

```
        d. Επέλεξε με τυχαίο τρόπο κάποια άτομα του πληθυσμού και
           εκτέλεσε πάνω σε αυτά μετάλλαξη χρησιμοποιώντας κάποια
           καλά ορισμένη μέθοδο.
// Έξοδος
   • η λύση η οποία έχει λάβει την καλύτερη βαθμολογία στον πληθυσμό
     την χρονική στιγμή τερματισμού της επανάληψης του αλγορίθμου.
```

Μπορούμε να δούμε γραφικά τον παραπάνω αλγόριθμο στην εικόνα που ακολουθεί :



Εικόνα 1 – γενική μορφή ενός εξελικτικού αλγορίθμου .

Υπάρχουν πολλά επιμέρους είδη εξελικτικών αλγορίθμων . Τα σημαντικότερα από αυτά είναι τα εξής (Streichert 1981) :

- **Γενετικοί Αλγόριθμοι :** οι Γενετικοί Αλγόριθμοι (ΓΑ) αποτελούν την πλέον περισσότερο γνωστή κατηγορία εξελικτικών αλγορίθμων . Οι ΓΑ στηρίζονται στις αρχές της εξελικτικής θεωρίας την οποία πρότεινε ο Κάρολος Δαρβίνος (Holland 1992) . Η δομή ενός ΓΑ είναι ακριβώς η δομή που περιγράφηκε παραπάνω χωρίς καμία παραλλαγή . Χαρακτηριστικό των ΓΑ είναι η αναπαράσταση κάθε πιθανής λύσης του πεδίου του προβλήματος με μία ισοδύναμη μορφή , την οποία ο ΓΑ μπορεί να χειριστεί

ευκολότερα . Η μορφή αυτή είναι συνήθως ένα αλφαριθμητικό , το οποίο περιλαμβάνει κυρίως αριθμούς . Είναι επίσης αρκετά συνηθισμένο οι αριθμοί αυτοί να είναι δυαδικοί .

- **Γενετικός Προγραμματισμός :** ο Γενετικός Προγραμματισμός αποτελεί μία παραλλαγή των Γενετικών Αλγορίθμων . Στον Γενετικό Προγραμματισμό οι πιθανές λύσεις του πεδίου του προβλήματος αναπαρίστανται ως προγράμματα υπολογιστών . Η δε αξιολόγηση της κάθε λύσης εξαρτάται από την ικανότητα του προγράμματος να λύσει αποδοτικά ένα υπολογιστικό πρόβλημα .
- **Εξελικτικός Προγραμματισμός :** ο Εξελικτικός Προγραμματισμός αποτελεί μία παραλλαγή του Γενετικού Προγραμματισμού . Η διαφορά του έγκειται στο ότι τα προγράμματα τα οποία εξελίσσονται έχουν συγκεκριμένη δομή και το μόνο το οποίο μπορεί να μεταβληθεί είναι οι τιμές των παραμέτρων τους .
- **Εξέλιξη με την χρήση Τεχνητών Νευρωνικών Δικτύων :** η τεχνική αυτή είναι παρόμοια με τον Γενετικό προγραμματισμό . Οι πιθανές λύσεις του πεδίου του προβλήματος αναπαρίστανται ως Τεχνητά Νευρωνικά Δίκτυα .
- **Εξελικτικές Στρατηγικές :** οι Εξελικτικές Στρατηγικές αναπαριστούν τις πιθανές λύσεις του πεδίου του προβλήματος με διανύσματα πραγματικών αριθμών .
- **Βελτιστοποίηση Νοημοσύνης Σμήνους (Particle Swarm Optimization) :** πρόκειται για μία πολύ δημοφιλή κατηγορία εξελικτικών αλγορίθμων . Ο αλγόριθμος PSO προσομοιώνει την κίνηση και συμπεριφορά των ζωικών ειδών τα οποία κινούνται σε σμήνη ή κοπάδια .
- **Βελτιστοποίηση Αποικίας Μυρμηγκιών (Ant Colony Optimization) :** η συγκεκριμένη κατηγορία εξελικτικών αλγορίθμων προσομοιώνει την κίνηση ενός αριθμού μυρμηγκιών τα οποία προσπαθούν να βρουν τροφή σε έναν χώρο . Χρησιμοποιείται ευρέως για την προσέγγιση της βέλτιστης λύσης ενός προβλήματος TSP (εύρεση του κύκλου σε έναν γράφο ο οποίος περνά από όλες τις κορυφές από ακριβώς μία φορά και έχει το μικρότερο συνολικό κόστος) .
- **Βελτιστοποίηση Αποικίας Μελισσών (Bee Colony Optimization) :** η συγκεκριμένη κατηγορία εξελικτικών αλγορίθμων προσομοιώνει την λειτουργία μίας κοινωνίας μελισσών για την εύρεση τροφής .
- **Βελτιστοποίηση Κοπαδιού Εικονικών Ψαριών (Artificial Fish Swarm Optimization) :** η συγκεκριμένη κατηγορία εξελικτικών αλγορίθμων προσομοιώνει την κίνηση ενός κοπαδιού ψαριών το οποίο κινείται στην θάλασσα με σκοπό την εύρεση μιας περιοχής πλούσιας σε τροφή .

2.2 Οι Γενετικοί Αλγόριθμοι

Οι Γενετικοί Αλγόριθμοι (ΓΑ) αποτελούν την πιο αντιπροσωπευτική κατηγορία Εξελικτικών Αλγορίθμων . Οι ΓΑ προτάθηκαν αρχικά από τους J. Holland (Holland 1992) και D. Goldberg (Goldberg 1989) και όπως αναφέρθηκε σκοπός τους είναι η προσομοίωση της εξελικτικής θεωρίας των ειδών για την προσέγγιση της βέλτιστης λύσης ενός προβλήματος . Ο ΓΑ αποτελεί στην ουσία μία επαναληπτική διαδικασία στην οποία εκτελούνται οι ενέργειες της αρχικοποίησης , αξιολόγησης, επιλογής , διασταύρωσης και μετάλλαξης ενός πληθυσμού πιθανών λύσεων . Έχει επικρατήσει η ορολογία γενιά (generation) για να περιγράψει την κάθε επανάληψη του ΓΑ . Στις επόμενες παραγράφους θα αναλύσουμε την κάθε μία από τις ενέργειες που λαμβάνουν χώρα σε κάθε γενιά και θα περιγράψουμε την λειτουργία τους .

2.2.1 Αναπαράσταση των πιθανών λύσεων

Πριν ξεκινήσουμε με την περιγραφή των βημάτων ενός ΓΑ , θα πρέπει πρώτα να περιγράψουμε το πως ο ΓΑ αναπαριστά τις πιθανές λύσεις του χώρου αναζήτησης . Οι ΓΑ και γενικά οι Εξελικτικοί Αλγόριθμοι αποτελούν τεχνικές επίλυσης προβλημάτων οι οποίες μπορούν θεωρητικά να εφαρμοστούν σε κάθε πρόβλημα . Όπως είναι φυσικό , το κάθε πρόβλημα είναι διαφορετικό από τα υπόλοιπα και οι πιθανές λύσεις ενός προβλήματος μπορεί να έχουν εντελώς άλλη μορφή από ότι ενός άλλου . Συνεπώς οι ΓΑ κωδικοποιούν τις πιθανές λύσεις σε μία μορφή τέτοια έτσι ώστε να μπορούν να εκτελέσουν εύκολα πάνω τους τις ενέργειες της

διασταύρωσης και της μετάλλαξης . Οι κωδικοποιημένες πιθανές λύσεις αποτελούνται από δομικά στοιχεία τα οποία ονομάζονται γονίδια (gene) . Η πιο συνηθισμένη μορφή κωδικοποίησης είναι η δυαδική κωδικοποίηση , στην οποία κάθε πιθανή λύση αναπαρίσταται ως μία δυαδική συμβολοσειρά και τα γονίδιά της είναι δυαδικά ψηφία . Θα επιδείξουμε τον τρόπο κωδικοποίησης των πιθανών λύσεων με την χρήση δυαδικών συμβολοσειρών , υποθέτοντας ότι ο χώρος αναζήτησης αποτελείται από διανύσματα πραγματικών αριθμών . Αυτή η μορφή του χώρου αναζήτησης δεν αποτελεί τον κανόνα αλλά είναι αρκετά συνηθισμένη (Streichert 2002) .

Ας υποθέσουμε λοιπόν ότι ο χώρος αναζήτησης PS αποτελείται από πραγματικά διανύσματα διάστασης n , της μορφής $(x_1, x_2, \dots, x_n)$. Υποθέτουμε επίσης ότι το κάθε στοιχείο x_i του διανύσματος παίρνει τιμές από ένα διάστημα $[d_i, u_i]$, όπου d_i το κάτω άκρο του διαστήματος και u_i το άνω άκρο . Υποθέτουμε επίσης ότι το στοιχείο x_i δεν μπορεί να πάρει άπειρες τιμές μέσα στο διάστημα αυτό αλλά οι τιμές αυτές μεταβάλλονται με σταθερό βήμα μεταβολής β_i . Έτσι λοιπόν αν x_i' και x_i'' είναι δύο συνεχόμενες τιμές του στοιχείου x_i μέσα στο διάστημα $[d_i, u_i]$, ισχύει ότι $x_i'' - x_i' = \beta_i$. Με την μοντελοποίηση αυτή , οι δυνατές τιμές κάθε στοιχείου x_i είναι συγκεκριμένες στον αριθμό και όχι άπειρες . Έτσι λοιπόν τα διανύσματα του χώρου αναζήτησης δεν είναι άπειρα στον αριθμό αλλά φράσσονται από κάποιο άνω όριο . Αυτό πρακτικά δεν αλλάζει την δομή του προβλήματος , ειδικά αν το βήμα μεταβολής είναι πολύ μικρό , αλλά είναι αναγκαίο για το επόμενο βήμα .

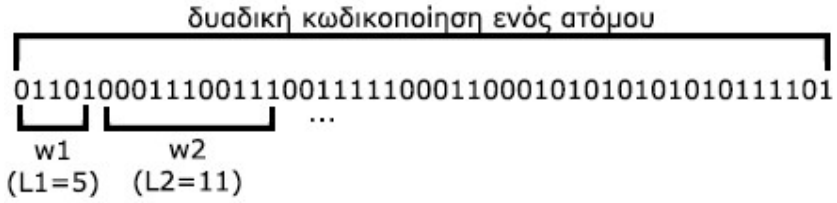
Το κάθε στοιχείο x_i του διανύσματος κωδικοποιείται με μία δυαδική συμβολοσειρά w_i , η οποία περιλαμβάνει L_i στον αριθμό δυαδικά ψηφία . Ο αριθμός των δυαδικών ψηφίων L_i είναι ο μικρότερος ακέραιος αριθμός για τον οποίο ισχύει ότι :

$$(u_i - d_i) \cdot 10^{\beta_i} \leq 2^{L_i} - 1$$

Έτσι λοιπόν το συνολικό μήκος L της δυαδικής συμβολοσειράς του διανύσματος θα είναι :

$$L = \sum_{i=1}^{n} L_i$$

Η παράθεση των δυαδικών συμβολοσειρών w_1, w_2, …, w_n οι οποίες αντιστοιχούν στα στοιχεία x_1, x_2, …, x_n συνθέτουν την τελική συμβολοσειρά και την συνολική κωδικοποίηση του διανύσματος $(x_1, x_2, \dots, x_n)$ (εικόνα 2) .



Εικόνα 2 – παράδειγμα δυαδικής κωδικοποίησης ενός ατόμου με L_1= 5 και L_2= 11 .

Η σχέση η οποία συνδέει κάθε στοιχείο x_i με την δυαδική του αναπαράσταση w_i είναι η εξής:

$$x_i = o_i + \frac{u_i - o_i}{2^{L_i} - 1} \cdot \left(\sum_{j=1}^{L} w_{i,(L_i - j + 1)} \cdot 2^{j-1} \right)$$

όπου $w_{i,n}$ το n-οστό δυαδικό ψηφίο της συμβολοσειράς w_i . Στον παραπάνω τύπο επίσης υποθέτουμε ότι η αρίθμηση των δυαδικών ψηφίων της κάθε συμβολοσειράς ξεκινά από το 0 . Οι παραπάνω τύποι είναι κατάλληλη στην περίπτωση όπου ο χώρος αναζήτησης του προβλήματος αποτελείται από πραγματικά διανύσματα οποιασδήποτε διάστασης . Φυσικά αν η μορφή των πιθανών λύσεων είναι διαφορετική , τότε η μετατροπή τους σε δυαδικές συμβολοσειρές θα πρέπει να γίνει με διαφορετικό τρόπο , ο οποίος ανά περίπτωση ίσως είναι λιγότερο ή περισσότερο περίπλοκος .

2.2.2 Εύρεση των ατόμων του αρχικού πληθυσμού

Το πρώτο βήμα της εκτέλεσης του ΓΑ είναι η εύρεση των ατόμων τα οποία θα αποτελέσουν τον αρχικό πληθυσμό . Όπως αναφέρθηκε , ο πληθυσμός P αποτελεί υποσύνολο του χώρου αναζήτησης PS και μάλιστα το μέγεθός του θα πρέπει να είναι αρκετά μικρότερο έναντι του συνολικού χώρου αναζήτησης . Ο περισσότερο συνηθισμένος τρόπος αρχικοποίησης είναι αυτή να γίνει με τυχαίο τρόπο . Αν υποθέσουμε ότι ο πληθυσμός περιλαμβάνει N στον αριθμό άτομα $\{a_1, a_2, \dots, a_N\}$ και το μήκος της δυαδικής τους αναπαράστασης είναι L , τότε ένας απλός αλγόριθμος αρχικοποίησης είναι ο εξής :

```
Για κάθε άτομο a_i του πληθυσμού P επανάλαβε
   Για κάθε δυαδικό ψηφίο j της συμβολοσειράς w_i του ατόμου a_i επανάλαβε
      Επέλεξε έναν τυχαίο αριθμό r στο διάστημα (0,1)
      Αν r < 0.5 τότε
         w_{i,j} = 0
      Αλλιώς
         w_{i,j} = 1
      Τέλος_Αν
   Τέλος_επανάληψης
Τέλος_επανάληψης
```

Ο παραπάνω αλγόριθμος παράγει παντελώς τυχαία άτομα και δεν δίνει καμία εγγύηση για την ποιότητά τους . Μπορούμε να τροποποιήσουμε τον αλγόριθμο με τρόπο τέτοιο έτσι ώστε το κάθε άτομο a_i να έχει τιμή αξιολόγησης f_i τουλάχιστον ίση με E (κάτω όριο) . Ο αλγόριθμος αρχικοποίησης τροποποιείται ως εξής :

```
Για κάθε άτομο a_i του πληθυσμού P επανάλαβε
   Επανάλαβε
      Για κάθε δυαδικό ψηφίο j της συμβολοσειράς w_i του ατόμου a_i επανάλαβε
         Επέλεξε έναν τυχαίο αριθμό r στο διάστημα (0,1)
         Αν r < 0.5 τότε
            w_{i,j} = 0
         Αλλιώς
            w_{i,j} = 1
         Τέλος_Αν
      Τέλος_επανάληψης
   Όσο f_i < E
Τέλος_επανάληψης
```

2.2.3 Αξιολόγηση των ατόμων του πληθυσμού

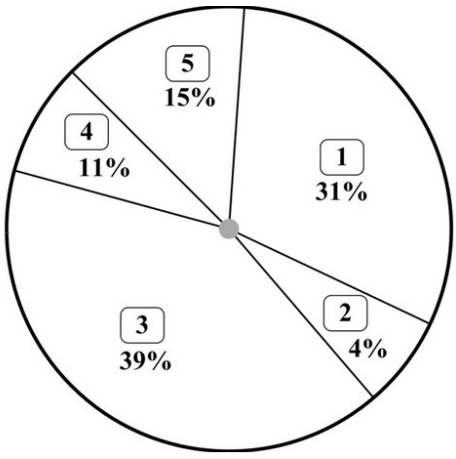
Η ποιότητα κάθε ατόμου του πληθυσμού θα πρέπει να αξιολογηθεί με την χρήση μιας συνάρτησης αξιολόγησης f . Άλλες ονομασίες για την συνάρτηση αξιολόγησης είναι αντικειμενική συνάρτηση ή συνάρτηση καταλληλότητας . Όπως αναφέρθηκε προηγουμένως , η συνάρτηση αυτή θα πρέπει να προσδίδει θετικές τιμές στα ορίσματά της και να είναι ανάλογη της ποιότητας των ατόμων . Αυτό πρακτικά σημαίνει ότι όσο ποιοτικότερο είναι ένα άτομο του πληθυσμού τόσο θα αυξάνει και η τιμή της αντικειμενικής συνάρτησης . Η αντικειμενική συνάρτηση συνήθως λαμβάνει ως όρισμα την κανονική μορφή του ατόμου και όχι την κωδικοποίησή του , χωρίς όμως να αποκλείεται το αντίθετο . Ένα χαρακτηριστικό που επίσης θα πρέπει να πληροί η συνάρτηση αξιολόγησης είναι να είναι εύκολα υπολογίσιμη . Πολύπλοκες συναρτήσεις αξιολόγησης απαιτούν συνήθως αρκετά μεγάλο υπολογιστικό χρόνο και αυξάνουν τον συνολικό χρόνο εκτέλεσης του ΓΑ .

2.2.4 Διαδικασία επιλογής

Επόμενο βήμα του ΓΑ είναι η διαδικασία της επιλογής . Η διαδικασία αυτή προσπαθεί να προσομοιώσει την φυσική επιλογή της εξελικτικής θεωρίας . Σύμφωνα με την Δαρβινική θεωρία της εξέλιξης , τα άτομα ικανότερα άτομα σε έναν πληθυσμό έχουν περισσότερες πιθανότητες επιβίωσης έναντι των αδυνάμων . Η περισσότερο συνηθισμένη διαδικασία επιλογής είναι ο γενετικός τελεστής της ρουλέτας (roulette wheel selection) . (Streichert 2002) . Σύμφωνα με τον τελεστή αυτόν , κάθε άτομο λαμβάνει μία πιθανότητα επιλογής (selection probability) η οποία είναι ανάλογη της τιμής αξιολόγησής τους . Έτσι λοιπόν όσο μεγαλύτερη είναι η καταλληλότητα ενός ατόμου τόσο αυξάνεται η πιθανότητα επιλογής του . Η πιθανότητα επιλογής p_i ενός ατόμου x_i είναι ο λόγος της καταλληλότητας του f_i σε σχέση με την συνολική καταλληλότητα του πληθυσμού :

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_i}$$

όπου N το μέγεθος του πληθυσμού . Μετά τον υπολογισμό της πιθανότητας επιλογής κάθε ατόμου , δημιουργείται ένας νέος πληθυσμός P' με βάση τον ήδη υπάρχων πληθυσμό P . Με την χρήση των πιθανοτήτων επιλογής δημιουργούμε μία μη δίκαιη ρουλέτα , όπου το εμβαδό του κάθε κελιού είναι ανάλογο της πιθανότητας επιλογής (εικόνα 3) . Για την δημιουργία του νέου πληθυσμού η ρουλέτα « περιστρέφεται » N στον αριθμό φορές και κάθε φορά επιλέγεται και από ένα άτομο . Είναι προφανές ότι όσο μεγαλύτερο είναι το εμβαδό του κελιού τόσο αυξάνει και η πιθανότητα συμμετοχής του ατόμου στο νέο πληθυσμό .



Εικόνα 3 – παράδειγμα ρουλέτας στην οποία το άτομο 3 έχει την μεγαλύτερη πιθανότητα επιλογής.

Για να υλοποιήσουμε αλγοριθμικά τον αλγόριθμο της ρουλέτας θα πρέπει να υπολογίσουμε επίσης και τις λεγόμενες συσσωρευμένες πιθανότητες . Η συσσωρευμένη πιθανότητα q_i ενός ατόμου a_i είναι ίση με το άθροισμα των πιθανοτήτων επιλογής από το πρώτο άτομο έως το άτομο a_i δηλαδή :

$$q_i = \begin{cases} p_1, & i = 1 \\ \sum_{i=1}^{i} p_i, & i > 1 \end{cases}$$

Έπειτα , για N στον αριθμό φορές παράγεται ένας τυχαίος πραγματικός αριθμός r στο διάστημα $(0,1)$ και επιλέγεται για το νέο πληθυσμό το άτομο a_i εκείνο για το οποίο ισχύει ότι $q_{i-1} < r \leq q_i$.

Αλγοριθμικά η διαδικασία της επιλογής είναι η εξής :

```
// Υπολογισμός των πιθανοτήτων επιλογής και των συσσωρευμένων πιθανοτήτων
Για i από 1 έως N επανάλαβε
   p_i ≔ f_i / Σ_{j=1}^{N} f_j
   Αν i = 1 τότε
      q_i ≔ p_1
   Αλλιώς
      q_i ≔ Σ_{j=1}^{i} p_j
   Τέλος_Αν
Τέλος_Επανάληψης
// Δημιουργία του νέου πληθυσμού
Για i από 1 έως N επανάλαβε
   Επέλεξε έναν τυχαίο αριθμό r στο διάστημα (0,1)
   Βρες το άτομο a_i για το οποίο ισχύει ότι q_{i-1} < r ≤ q_i και πρόσθεσέ το
    στο νέο πληθυσμό P'
Τέλος_επανάληψης
```

Είναι προφανές από τον παραπάνω αλγόριθμο στο νέο πληθυσμός P' να έχει επιλεγεί το ίδιο άτομο περισσότερες από μία φορές . Ο παραπάνω αλγόριθμος προσομοιώνει την διαδικασίας της φυσική επιλογής στην φύση αφού δίνει μεγαλύτερες πιθανότητες « επιβίωσης » στα καταλληλότερα άτομα χωρίς όμως να αποκλείει εντελώς τα αδύναμα , αφού η πιθανότητα επιλογής ενός ατόμου με μικρή τιμή αξιολόγησης είναι μεν μικρή αλλά υπαρκτή . Στην συντριπτική πλειοψηφία των περιπτώσεων , η μέση τιμή αξιολόγησης του πληθυσμού P' είναι καλύτερη της μέσης τιμής αξιολόγησης του πληθυσμού P .

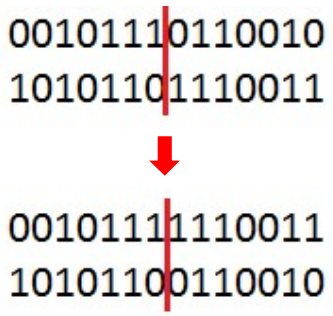
2.2.5 Διαδικασία διασταύρωσης

Η διαδικασία της διασταύρωσης αποτελεί ίσως τον σημαντικότερο γενετικό τελεστή εκ των επιλογής , διασταύρωσης και μετάλλαξης καθώς είναι αυτή η οποία κάνει τον ΓΑ να εντοπίσει νέες πιθανές λύσεις στον χώρο αναζήτησης . Η διαδικασία της διασταύρωσης εκτελείται στα άτομα του πληθυσμού P' και δημιουργεί ένα νέο πληθυσμό P'' . Η μέθοδος διασταύρωσης εφαρμόζεται στην κωδικοποιημένη μορφή των ατόμων του πληθυσμού η οποία όπως είδαμε είναι συνήθως η δυαδική κωδικοποίηση . Στο σημείο αυτό θα παρουσιάσουμε τον συνηθέστερο τρόπο διασταύρωσης , όταν χρησιμοποιείται η δυαδική κωδικοποίηση , ο οποίος δεν είναι άλλος από την διασταύρωση μονού σημείου .

Ο τελεστής της διασταύρωσης μονού σημείου απαιτεί την δημιουργία ζευγών γονέων έτσι ώστε να μπορεί να δημιουργήσει τους απογόνους . Για κάθε ζεύγος γονέων προκύπτει και ένας ζεύγος απογόνων , τα οποία αντικαθιστούν τους γονείς τους στον πληθυσμό P'' . Τα ζεύγη

γονέων επιλέγονται με τυχαίο τρόπο από το σύνολο P' . Ειδικότερα , για την επιλογή των ατόμων που θα διασταυρωθούν ο τελεστής διασταύρωσης χρησιμοποιεί μία σταθερά $P_c < 1$, η οποία ονομάζεται πιθανότητα διασταύρωσης . Για κάθε άτομο του πληθυσμού ο αλγόριθμος επιλέγει έναν τυχαίο αριθμό r στο διάστημα $(0,1)$ και ελέγχει αν ο αριθμός είναι μικρότερος της P_c . Αν αυτό ισχύει , τότε το άτομο επιλέγεται για διασταύρωση ενώ σε διαφορετική περίπτωση όχι . Για να είναι δυνατή η δημιουργία των ζευγών, θα πρέπει ο αριθμός των ατόμων που επιλέχθηκαν για διασταύρωση να είναι άρτιος . Αν είναι περιττός τότε ο αλγόριθμος επιλέγει με τυχαίο τρόπο ένα ακόμη άτομο από το πληθυσμό P' και είναι αρκετά σύνηθες το άτομο που επιλέγεται να είναι αυτό με την μεγαλύτερη καταλληλότητα .

Στην συνέχεια για κάθε ζεύγος εκτελείται η διασταύρωση συναρτήσει του μήκους των δυαδικών συμβολοσειρών . Ειδικότερα, αν το μήκος των δυαδικών συμβολοσειρών είναι ίσο με L τότε ανάμεσα από τα δυαδικά ψηφία υπάρχουν $L-1$ ισοπίθανα σημεία διασταύρωσης . Ο τελεστής διασταύρωσης επιλέγει τυχαία ένα από τα σημεία αυτά, έστω το K . Οι απόγονοι διατηρούν τα δυαδικά ψηφία των γονέων τους έως το σημείο διασταύρωσης K αλλά από το σημείο αυτό και μετά ανταλλάσσουν τα δυαδικά τους ψηφία (εικόνα 4) . Όσα άτομα του πληθυσμού P' δεν επιλέχθηκαν για διασταύρωση μένουν αναλλοίωτα στον πληθυσμό P'' που προκύπτει μετά την διασταύρωση .



Εικόνα 4 – παράδειγμα τελεστή διασταύρωσης μονού σημείου.

Αλγοριθμικά η διαδικασία της διασταύρωσης μονού σημείου είναι η εξής :

```
// Σύνολο το οποίο αποθηκεύει τα άτομα προς διασταύρωση
Parents ≔ { }
// Εύρεση των ατόμων προς διασταύρωση
Για i από 1 έως N επανάλαβε
   Επέλεξε έναν τυχαίο αριθμό r στο διάστημα (0,1)
   Αν r < P_c τότε
      Πρόσθεσε το άτομο a_i στο σύνολο Parents
   Αλλιώς
      Πρόσθεσε το άτομο a_i στο νέο πληθυσμό P''
   Τέλος_Αν
Τέλος_Επανάληψης
// Έλεγχος αριθμού γονέων
Αν πληθάρισμος |Parents| είναι περιττός τότε
   Επέλεγε το στοιχείο a_Best (με την μεγαλύτερη καταλληλότητα)
   από το σύνολο P' και πρόσθεσέ το στο σύνολο Parents
Τέλος_Αν
// Εκτέλεση των διασταυρώσεων
Για i από 1 έως |Parents|/2 επανάλαβε
   Επέλεξε τυχαία δύο γονείς P_x και P_y από το σύνολο Parents
   Επέλεξε τυχαία ένα σημείο διασταύρωσης K από τα L − 1 ισοπίθανα
```

```
    σημεία διασταύρωσης
    Δημιούργησε τους απογόνους O_x και O_y κρατώντας αμετάβλητο τα τμήματα των
    γονέων πριν το σημείο διασταύρωσης K και ανταλλάσσοντας τα τμήματα μετά
    το σημείο αυτό
    Διέγραψε τους γονείς P_x και P_y από το σύνολο Parents και πρόσθεσε τους
    απογόνους O_x και O_y στο νέο πληθυσμό P''
Τέλος_επανάληψης
```

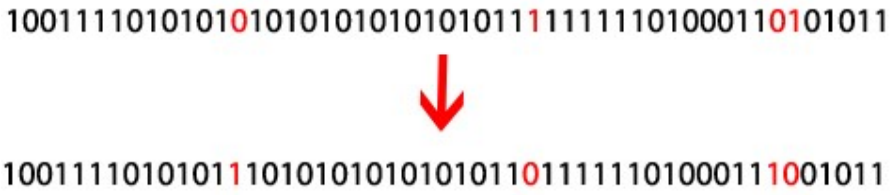
2.2.6 Διαδικασία μετάλλαξης

Όπως αναφέρθηκε προηγουμένως , η διαδικασία της μετάλλαξης έχει δύο διακριτούς στόχους :

1. Να απεμπλέξει τον αλγόριθμο τον αλγόριθμο από τοπικά βέλτιστα στα οποία είναι πιθανό να βρεθεί . Το συγκεκριμένο πρόβλημα (παγίδευση σε κάποιο τοπικό βέλτιστο) αποτελεί ένα από τα μείζοντα προβλήματα των εξελικτικών τεχνικών γενικότερα .
2. Να κάνει τον αλγόριθμο να επισκεφθεί περιοχές του χώρου αναζήτησης τις οποίες ο αλγόριθμος δεν είναι σε θέση να προσεγγίσει με την χρήση του τελεστή διασταύρωσης . Για παράδειγμα μπορεί να υπάρχουν πιθανές λύσεις του χώρου αναζήτησης οι οποίες δεν γίνεται να παραχθούν ποτέ από τον τελεστή της διασταύρωσης .

Η διαδικασία της μετάλλαξης εφαρμόζεται για κάθε άτομο του πληθυσμού P'' , ο οποίος προκύπτει μετά την διασταύρωση . Η διαδικασία της μετάλλαξης προσομοιώνει στην ουσία τη μετάλλαξη η οποία λαμβάνει χώρα στη φύση . Ειδικότερα , σε κάποια από τα άτομα ενός φυσικού πληθυσμού, τα χαρακτηριστικά του γενετικού τους υλικού μεταβάλλονται με τυχαίο τρόπο . Η αλλαγή αυτή πάνω στα γενετικά τους χαρακτηριστικά μπορεί είτε να έχει ευεργετικές επιπτώσεις στα άτομα αυτά , κάνοντάς τα καλύτερα , ενώ φυσικά σε κάποιες περιπτώσεις οι επιπτώσεις μπορεί να είναι πολύ δυσάρεστες .

Στους ΓΑ η μετάλλαξη εκτελείται συνήθως στην κωδικοποιημένη μορφή των ατόμων του πληθυσμού . Στην περίπτωση που η κωδικοποίηση η οποία χρησιμοποιείται είναι η δυαδική , ο τελεστής μετάλλαξης διατρέχει ένα-ένα τα δυαδικά ψηφία για κάθε άτομο του πληθυσμού . Για κάθε δυαδικό ψηφίο επιλέγει έναν τυχαίο αριθμό r στο διάστημα $(0,1)$. Αν ο τυχαίος αυτός αριθμός είναι μικρότερος από μία πιθανότητα μετάλλαξης P_m , τότε το αντίστοιχο δυαδικό ψηφίο αντιστρέφεται . Είναι σαφές ότι η πιθανότητα μετάλλαξης P_m θα πρέπει να είναι αρκετά μικρή , έτσι ώστε ο ΓΑ να μην εκφυλλιστεί σε τυχαία αναζήτηση . Θα συμβολίσουμε τον πληθυσμό ο οποίος προκύπτει μετά την μετάλλαξη ως P''' . Ένα παράδειγμα μετάλλαξης σε ένα άτομο με δυαδική κωδικοποίηση παρουσιάζεται στην εικόνα 5 .



Εικόνα 5 – παράδειγμα μετάλλαξης σε άτομο με δυαδική κωδικοποίηση .

Αλγοριθμικά η διαδικασία της μετάλλαξης είναι η παρακάτω .

```
// Για κάθε άτομο του πληθυσμού που έχει προκύψει μετά την διασταύρωση
Για κάθε άτομο a_i του πληθυσμού P'' επανάλαβε
   // Διατρέχουμε το κάθε δυαδικό ψηφίο του ατόμου
   Για j από 1 έως L επανάλαβε
      Επέλεξε έναν τυχαίο αριθμό r στο διάστημα (0,1)
      // Ελέγχουμε αν το δυαδικό ψηφίο θα μεταλλαχθεί
```

```
      Αν r < P_m τότε
         // Σε περίπτωση μετάλλαξης το αντιστρέφουμε
         Αν a_{i,j} = 1 τότε
            a_{i,j} := 0
         Αλλιώς
            a_{i,j} := 1
         Τέλος_Αν
      Τέλος_Αν
   Τέλος_Επανάληψης
   Πρόσθεσε το άτομο a_i στον πληθυσμού P'''
Τέλος_Επανάληψης
```

Ένα όχι και τόσο σπάνιο φαινόμενο είναι αυτό της παραγωγής μη έγκυρων ατόμων του πληθυσμού , τα οποία μπορεί να προκύψουν από τον τελεστή της μετάλλαξης . Προηγουμένως είδαμε ότι υπάρχει συγκεκριμένος τρόπος αντιστοίχισης ενός ατόμου a_i σε μία δυαδική συμβολοσειρά w_i μήκους L . Αυτό φυσικά δεν σημαίνει ότι κάθε δυαδική συμβολοσειρά αντιστοιχίζεται σε ένα έγκυρο άτομο . Έτσι λοιπόν υπάρχει περίπτωση μία δυαδική συμβολοσειρά η οποία θα προκύψει από μία μετάλλαξη να αντιστοιχεί σε άτομο το οποίο δεν είναι έγκυρο με αποτέλεσμα ο ΓΑ να πρέπει να προβεί σε διορθωτική κίνηση .

2.2.7 Κριτήριο τερματισμού

Το κριτήριο τερματισμού του ΓΑ , όπως και κάθε εξελικτικού αλγορίθμου , μπορεί να διαφέρει ανά περίπτωση . Μία λίστα με συνηθέστερα κριτήρια τερματισμού παρουσιάζεται παρακάτω .

- Ο ΓΑ να εκτελεστεί σε έναν προκαθορισμένο αριθμό επαναλήψεων . Το κριτήριο αυτό κάνει την υπόθεση ότι ο ΓΑ έχει εξελίξει τα άτομα του πληθυσμού του σε σημείο τέτοιο έτσι ώστε κάποιο από τα άτομα αυτά να βρίσκεται αρκετά κοντά στο ολικό βέλτιστο του χώρου αναζήτησης . Ο αριθμός των επαναλήψεων ο οποίος είναι απαραίτητος εξαρτάται από το είδος του προβλήματος , στο οποίο ο ΓΑ εφαρμόζεται και από το μέγεθος του χώρου αναζήτησης .
- Η μέση καταλληλότητα των ατόμων του πληθυσμού να μην αλλάζει σημαντικά από γενιά σε γενιά . Ειδικότερα , μπορούμε να ορίσουμε ως μέση καταλληλότητα μίας γενιάς t την αναμενόμενη τιμή (δηλαδή τον μέσο όρο) των τιμών αξιολόγησης των ατόμων του πληθυσμού . Έτσι λοιπόν , για έναν πληθυσμό P μεγέθους N η μέση καταλληλότητα $\bar{F}$ ισούται με :

$$\overline{F(t)} = \frac{\sum_{i=1}^{N} f_i(t)}{N}$$

όπου $f_i(t)$ η τιμή αξιολόγησης του ατόμου x_i την χρονική στιγμή t . Συνεπώς αν για δύο διαδοχικές γενιές t και $t+1$ η μεταβολή της $\bar{F}$ είναι πολύ μικρή (δεν ξεπερνά ένα κατώφλι ε) τότε ο ΓΑ δεν έχει νόημα να συνεχίσει :

$$\left|\overline{F(t+1)} - \overline{F(t)}\right| < \varepsilon$$

- Ορισμός της μέγιστης τιμής καταλληλότητας , την οποία αν κάποιο άτομο του πληθυσμού λάβει τότε η εκτέλεση του ΓΑ μπορεί να τερματίσει . Ειδικότερα , μπορούμε να θεσπίσουμε ένα άνω όριο F_{max} , το οποίο θεωρούμε ότι αν κάποιο μέλος του πληθυσμού έχει ως τιμή ίση ή μεγαλύτερη αυτού τότε έχει πλησιάσει αρκετά το ολικό βέλτιστο του χώρου αναζήτησης . Φυσικά η εν λόγω προσέγγιση προϋποθέτει ότι ο σχεδιαστής του ΓΑ έχει καλή γνώση της δομής του χώρου αναζήτησης και είναι σε θέση να προσεγγίσει την τιμή αξιολόγησης της βέλτιστης δυνατής λύσης .

2.3 Ο αλγόριθμος PSO

Ο αλγόριθμος PSO (Particle Swarm Optimization) προτάθηκε από τους Kennedy και Ebberhart (Kennedy , Eberhart 1995) και αποτελεί μία από τις δημοφιλέστερες εξελικτικές τεχνικές . Έχει αποδειχθεί ότι ο αλγόριθμος PSO λειτουργεί ιδιαίτερα αποδοτικά σε πολύπλοκους χώρους αναζήτησης , των οποίων τα στοιχεία τους είναι διανύσματα μεγάλων διαστάσεων . Στο σημείο αυτό θα πρέπει να τονίσουμε ότι η μοντελοποίηση των πιθανών λύσεων ως διανύσματα (οποιασδήποτε διάστασης) είναι υποχρεωτική για την εκτέλεση του αλγορίθμου . Ο PSO έχει την φήμη της ταχύτερης , σε σχέση με τους υπόλοιπους εξελικτικούς αλγορίθμους , σύγκλισής του προς το ολικό βέλτιστο του χώρου αναζήτησης . Όπως είναι φυσικό όμως , αφού και αυτός αποτελεί μία μετα - ευριστική μέθοδο , δεν μπορεί να εγγυηθεί την επιστροφή της βέλτιστης λύσης αλλά μία καλή προσέγγιση αυτής .

Ο αλγόριθμος PSO προσομοιώνει την συμπεριφορά των ειδών του ζωικού βασιλείου τα οποία κινούνται σε σμήνη στο φυσικό τους περιβάλλον . Ένα χαρακτηριστικό παράδειγμα είναι τα αποδημητικά πουλιά τα οποία είναι σε θέση να μετακινούνται από ένα σημείο της Γης σε ένα άλλο , χωρίς να έχουν αναπτύξει κάποιον συγκεκριμένο τρόπο επικοινωνίας . Είναι επίσης συνηθισμένο πολλά είδη ψαριών να κινούνται σε κοπάδια και να μπορούν να εντοπίζουν περιοχές του ωκεανού με μεγάλη ποσότητα τροφής , κινούμενα με σταθερό τρόπο προς αυτές . Οι χαρακτηριστικές αυτές συμπεριφορές των ζωικών ειδών κίνησαν το ενδιαφέρον των Kennedy και Ebberhart , οι οποίοι θέλησαν να μοντελοποιήσουν την κίνηση ενός σμήνους ζώων σε κίνηση ενός σμήνους πιθανών λύσεων οι οποίες σταδιακά θα προσεγγίσουν το ολικό βέλτιστο του χώρου αναζήτησης .

Στον αλγόριθμο PSO ο πληθυσμός που εξελίσσεται ονομάζεται σμήνος (swarm) ενώ τα μέλη του ονομάζονται μόρια (particles) . Κάθε μόριο βρίσκεται σε κάθε χρονική στιγμή (γενιά) σε ένα σημείο του χώρου αναζήτησης . Ο αλγόριθμος αποθηκεύει επιπρόσθετα για κάθε μόριο το καλύτερο σημείο στο οποίο για την ώρα έχει βρεθεί (δηλαδή το σημείο στο οποίο το συγκεκριμένο μόριο έχει παρουσιάσει την μεγαλύτερη τιμή της συνάρτησης αξιολόγησης) . Το βέλτιστο προσωπικό σημείο του κάθε μορίου συμβολίζεται με p_{best} (personal best) . Εκτός αυτού , ο αλγόριθμος επίσης αποθηκεύει το καλύτερο σημείο στο οποίο έχει βρεθεί ο πληθυσμός συνολικά (στην ουσία δηλαδή το καλύτερο από τα σημεία p_{best}) . Το καλύτερο σημείο στο οποίο ο πληθυσμός έχει βρεθεί συμβολίζεται με g_{best} (global best) . Σε κάθε γενιά το κάθε μόριο μεταβάλλει την τρέχουσα θέση του και μεταβαίνει σε μία νέα . Η νέα θέση υπολογίζεται προσθέτοντας την τρέχουσα θέση (τρέχον διάνυσμα) με ένα διάνυσμα μεταβολής , το οποίο ονομάζεται ταχύτητα (velocity) . Η ταχύτητα υπολογίζεται λαμβάνοντας υπ' όψη 4 πράγματα :

1. την τρέχουσα τιμή του διανύσματος της ταχύτητας
2. την τρέχουσα θέση στην οποία το μόριο βρίσκεται
3. την θέση p_{best}
4. την θέση g_{best} .

Ειδικότερα , η λογική η οποία κρύβεται πίσω από τον υπολογισμό του διανύσματος της ταχύτητας είναι ιδιαίτερα απλή . Το μόριο έχει ήδη μία ταχύτητα, η οποία είναι στην ουσία μία ροπή προς μία κατεύθυνση στον χώρο αναζήτησης . Αν κάποιο μόριο έχει βρεθεί σε κάποιο πολύ καλό σημείο του χώρου αναζήτησης (g_{best}) τότε ειδοποιεί τα υπόλοιπα μόρια για την ύπαρξη του σημείου αυτού . Τα υπόλοιπα μόρια όμως δεν αγνοούν το γεγονός ότι και αυτά έχουν βρεθεί σε κάποιο καλό σημείο του χώρου αναζήτησης (p_{best}) οπότε ίσως κοντά στην θέση αυτή να υπάρχει κάποια καλύτερη , ακόμα και σε σχέση με την g_{best} . Συνεπώς το μόριο μεταβάλλει την ταχύτητά του λαμβάνοντας υπ' όψη τόσο το προς τα που κινείται ως τώρα όσο και τις πληροφορίες g_{best} όσο και p_{best} . Η παραπάνω εκτέλεση του αλγορίθμου PSO οδηγεί σταδιακά στην σύγκλιση όλων των μορίων του σμήνους γύρω από ένα g_{best} , το οποίο στην πλειοψηφία των περιπτώσεων είναι αρκετά κοντά στο ολικό βέλτιστο του χώρου αναζήτησης . Θα δώσουμε τον τυπικό ορισμό του αλγορίθμου PSO παρακάτω .

2.3.1 Τυπικός ορισμός του αλγορίθμου PSO

Έστω PS ένας πραγματικός χώρος αναζήτησης διάστασης n , με $PS \subseteq \mathcal{R}^n$. Έστω επίσης ένα υποσύνολο $S \subset PS$ του χώρου αναζήτησης το οποίο ονομάζεται σμήνος (swarm) . Το μέγεθος

του σμήνους είναι αρκετά μικρότερο από αυτό του χώρου αναζήτησης ($|S| \ll |PS|$). Ονομάζουμε κάθε στοιχείο $x_i \in S$ μόριο (particle) . Επειδή ο αλγόριθμος εκτελείται σε χρονικές στιγμές, η τιμή του κάθε μορίου $x_i \in S$ τη χρονική στιγμή t συμβολίζεται με $x_i(t)$. Θεωρούμε ότι η πρώτη χρονική στιγμή είναι η $t = 0$. Σε κάθε χρονική στιγμή το μόριο μεταβάλλει το διάνυσμα της τρέχουσας θέσης του κάνοντας χρήση ενός διανύσματος μεταβολής, το οποίο συμβολίζεται με $v_i(t)$ και ονομάζεται ταχύτητα (velocity) . Η μεταβολή της τρέχουσας θέσης $x_i(t)$ προκύπτει ως εξής (Valle et all 2008) :

$$\overrightarrow{x_i(t)} = \overrightarrow{x_i(t-1)} + \overrightarrow{v_i(t)}$$

Η ταχύτητα του κάθε μορίου αλλάζει και αυτή κάθε χρονική στιγμή και προκύπτει λαμβάνοντας υπ 'όψη 4 παραμέτρους :

1. την ταχύτητα που είχε το μόριο την προηγούμενη χρονική στιγμή , δηλαδή την $\overrightarrow{v_i(t-1)}$.
2. το διάνυσμα στο οποίο το μόριο είχε βρεθεί την προηγούμενη χρονική στιγμή , δηλαδή το $\overrightarrow{x_i(t-1)}$.
3. το καλύτερο διάνυσμα του χώρου αναζήτησης (αυτό με την μεγαλύτερη τιμή αξιολόγησης) στο οποίο το μόριο έχει βρεθεί ως την τρέχουσα χρονική στιγμή και το οποίο συμβολίζεται με $\overrightarrow{pbest_i}$.
4. το καλύτερο διάνυσμα του χώρου αναζήτησης στο οποίο έχει βρεθεί οποιοδήποτε μόριο του πληθυσμού ως την τρέχουσα χρονική στιγμή και το οποίο συμβολίζεται με $\overrightarrow{gbest}$.

Η ταχύτητα προκύπτει από τον παρακάτω τύπο (Valle et all 2008) :

$$\overrightarrow{v_i(t)} = \overrightarrow{v_i(t-1)} + \varphi_1 \cdot rand_1 \cdot \left(\overrightarrow{pbest_i} - \overrightarrow{x_i(t-1)}\right) + \varphi_2 \cdot rand_2 \cdot \left(\overrightarrow{gbest} - \overrightarrow{x_i(t-1)}\right)$$

όπου οι φ_1, φ_2 είναι δύο σταθερές μεγαλύτερες του μηδενός οι οποίες ονομάζονται σταθερές επιτάχυνσης ενώ οι $rand_1$, $rand_2$ δύο τυχαίες πραγματικές τιμές με πεδίο τιμών το διάστημα $[0,1]$. Μπορούμε να δούμε ότι ο τύπος της ταχύτητας αποτελείται ένα άθροισμα 3 διακριτών όρων (Valle et all 2008) :

1. Ο πρώτος όρος $\overrightarrow{v_i(t-1)}$ ονομάζεται συνήθεια (habbit) και μοντελοποιεί την τάση που έχει ένα μόριο του σμήνους να διατηρεί την ταχύτητα που έχει ως την τρέχουσα χρονική στιγμή (στην ουσία δηλαδή να συνεχίζει να κινείται με την φορά που ως τώρα είχε) .
2. Ο δεύτερος όρος $\varphi_1 \cdot rand_1 \cdot \left(\overrightarrow{pbest_i} - \overrightarrow{x_i(t-1)}\right)$ ονομάζεται μνήμη (memory) και μοντελοποιεί την τάση που έχει ένα μόριο του σμήνους να απομνημονεύει την καλύτερη θέση στην οποία έχει βρεθεί μέχρι την τρέχουσα χρονική στιγμή και την επιθυμία να κινηθεί ξανά προς αυτή . Η μετακίνηση προς τη θέση αυτή κανονικοποιείται κάνοντας χρήση των παραμέτρων φ_1 και $rand_1$.
3. Ο τρίτος όρος $\varphi_2 \cdot rand_2 \cdot \left(\overrightarrow{gbest} - \overrightarrow{x_i(t-1)}\right)$ ονομάζεται κοινωνική γνώση (social knowledge) και μοντελοποιεί την έλξη που έχει ένα μόριο του σμήνους προς την καλύτερη θέση που έχει ανακαλυφθεί από όλο το σμήνος την τρέχουσα χρονική στιγμή . Η μετακίνηση προς τη θέση αυτή κανονικοποιείται κάνοντας χρήση των παραμέτρων φ_2 και $rand_2$.

Σε μορφή ψευδοκώδικα , ο αλγόριθμος PSO παρουσιάζεται παρακάτω .

```
// Είσοδος
   • το πεδίο ορισμού του προβλήματος PS (χώρος αναζήτησης)
   • μία συνάρτηση ελέγχου της ποιότητας κάθε υποψήφιας λύσης f
     (συνάρτηση αξιολόγησης)
// Επεξεργασία
   1. Επέλεξε με κάποια διαδικασία (τυχαία ή μη) τα μόρια του αρχικού
      σμήνους S, το οποίο αποτελεί γνήσιο υποσύνολο του χώρου
      αναζήτησης PS (S ⊂ PS). Κάθε μόριο αντιστοιχίζεται σε ένα σημείο
      (πιθανή λύση) του χώρου αναζήτησης PS.
```

2. Υπολόγισε την τιμή αξιολόγησης του κάθε μορίου χρησιμοποιώντας την συνάρτηση αξιολόγησης f.
3. Για κάθε μόριο σύγκρινε την τρέχουσα τιμή αξιολόγησης με την τιμή αξιολόγησης της καλύτερης θέσης στην οποία το μόριο βρέθηκε ως τώρα ($pbest$). Αν η τρέχουσα τιμή αξιολόγησης είναι καλύτερη, τότε θέσε ως $pbest$την τρέχουσα θέση του μορίου.
4. Βρες την καλύτερη τιμή αξιολόγησης που υπάρχει στο σμήνος την τρέχουσα χρονική στιγμή, συγκρίνοντας όλα τα pbest μεταξύ τους. Αν η τιμή αυτή είναι καλύτερη από την τιμή αξιολόγησης της καλύτερης θέσης που έχει βρεθεί ως τώρα ($gbest$), ανανέωσε την θέση $gbest$.
5. Υπολόγισε το νέο διάνυσμα της ταχύτητας για κάθε μόριο με βάση τον παρακάτω τύπο:

$$\overrightarrow{v_i(t)} = \overrightarrow{v_i(t-1)} + \varphi_1 \cdot rand_1 \cdot \left(\overrightarrow{pbest_i} - \overrightarrow{x_i(t-1)}\right) + \varphi_2 \cdot rand_2 \cdot \left(\overrightarrow{gbest} - \overrightarrow{x_i(t-1)}\right)$$

6. Υπολόγισε τη νέα θέση κάθε μορίου με βάση τον παρακάτω τύπο:

$$\overrightarrow{x_i(t)} = \overrightarrow{x_i(t-1)} + \overrightarrow{v_i(t)}$$

7. Έλεγξε αν ικανοποιείται το κριτήριο τερματισμού. Αν όχι τότε πήγαινε στο βήμα 2.

Η κλασική μορφή του αλγορίθμου PSO που παρουσιάστηκε παραπάνω δεν είναι η μόνη δυνατή , καθώς υπάρχει μία πληθώρα παραλλαγών του . Ο αλγόριθμος PSO αποτελεί αντικείμενο εκτεταμένης έρευνας με στόχο τη συνεχή βελτιστοποίησή του και αύξηση της απόδοσής του . Τα βασικότερα αντικείμενα προς έρευνα παρουσιάζονται παρακάτω .

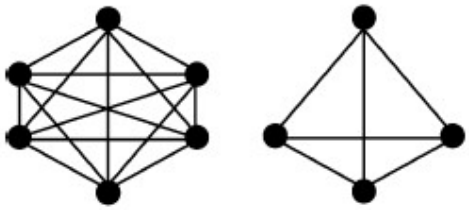
2.3.2 Αρχικοποίηση του σμήνους

Η ποιότητα των αρχικών μορίων του σμήνους διαδραματίζει σημαντικό ρόλο στην μετέπειτα εκτέλεση του αλγόριθμου PSO . Αν οι αρχικές θέσεις των μορίων λάβουν τιμές με όσο το δυνατό περισσότερο κατανεμημένο τρόπο στον χώρο αναζήτησης τότε οι πιθανότητες εξεύρεσης ενός καλού ολικού βέλτιστου είναι αυξημένες αρκετά . Αντίθετα , αν οι αρχικές θέσεις των μορίων επικεντρώνονται σε ένα μικρό κομμάτι του χώρου αναζήτησης , τότε ο κίνδυνος παγίδευσης του αλγορίθμου σε κάποιο τοπικό βέλτιστο αυξάνεται . Οι Richards και Ventura (Richards , Ventura 2004) έδειξαν πειραματικά ότι η αρχικοποίηση των μορίων με ομοιόμορφο τρόπο με βάση μια πιθανοτική κατανομή συμβάλλει σημαντικά στην ποιότητα του τελικού αποτελέσματος του αλγορίθμου , αφού αυξάνει την πιθανότητα για την κάλυψη μεγαλύτερου μέρους του χώρου αναζήτησης . Θα πρέπει πάντως να δοθεί ιδιαίτερη μέριμνα στον αλγόριθμο αρχικοποίησης του σμήνους , έτσι ώστε αυτός να μην είναι ιδιαίτερα απαιτητικός υπολογιστικά , καθώς ελλοχεύει ο κίνδυνος καθυστέρησης της εκτέλεσης του αλγορίθμου από το πρώτο βήμα του και μόνο .

2.3.4 Τοπολογία σμήνους

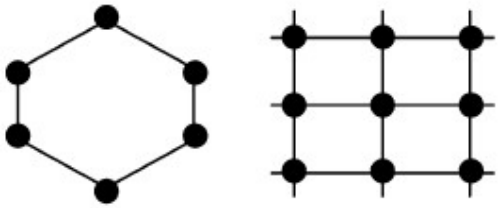
Με τον όρο τοπολογία σμήνους αναφερόμαστε στον τρόπο επικοινωνίας μεταξύ των μορίων του σμήνους . Ειδικότερα , στον κλασικό αλγόριθμο PSO που παρουσιάστηκε προηγουμένως , όλα τα μόρια επικοινωνούν μεταξύ τους , αφού υπάρχει ένα και μόνο σημείο $gbest$. Όταν ο αλγόριθμος PSO επιτρέπει την επικοινωνία μεταξύ όλων των μορίων του πληθυσμού λέμε ότι ακολουθεί τοπολογία πλήρους επικοινωνίας . Παράδειγμα τοπολογιών πλήρους επικοινωνίας παρουσιάζονται στην εικόνα 6 . Το μειονέκτημα με τις τοπολογίες πλήρους επικοινωνίας είναι ότι αυξάνουν τις πιθανότητες παγίδευσης του αλγορίθμου σε κάποιο τοπικό βέλτιστο , αφού όλα

τα μόρια του σμήνους κινούνται πάντα προς το ίδιο $gbest$. Το πλεονέκτημα δε είναι η γρήγορη σχετικά σύγκλιση του αλγορίθμου , αφού



Εικόνα 6 – τοπολογίες σμήνους πλήρους επικοινωνίας .

Μία παραλλαγή του κλασικού αλγορίθμου PSO είναι αυτός ο οποίος χρησιμοποιεί τοπολογίες μερικής επικοινωνίας . Σύμφωνα με την θεώρηση αυτή , τα μόρια του σμήνους δεν επικοινωνούν όλα μεταξύ τους αλλά το καθένα μπορεί να επικοινωνήσει με ένα υποσύνολο από τα υπόλοιπα . Παραδείγματα τοπολογιών μερικής επικοινωνίας παρουσιάζονται στην εικόνα 7 .



Εικόνα 7 – τοπολογίες σμήνους μερικής επικοινωνίας .

Στις τοπολογίες μερικής επικοινωνίας κάθε μόριο επικοινωνεί με έναν αριθμό άλλων μορίων τα οποία αποτελούν την γειτονιά του . Πλέον κάθε γειτονιά έχει το δικό της καλύτερο σημείο , το οποίο ονομάζεται $lbest$ (local best) . Φυσικά δεν υπάρχει κανένα μόριο το οποίο να είναι πλήρως απομονωμένο από το υπόλοιπο συνολικό σμήνος αφού η γειτονιά κάθε μορίου έχει μέγεθος τουλάχιστον 1 . Στην ουσία , σε μια τοπολογία σμήνους μερικής επικοινωνίας το αρχικό σμήνος διαιρείται σε μικρότερα υπο - σμήνη όπου το κάθε ένα από αυτά εξελίσσεται σε διαφορετικό τμήμα του χώρου αναζήτησης . Μετά το τέλος της εκτέλεσης του αλγορίθμου επιστρέφεται το $lbest$ του κάθε υπο-σμήνους και το καλύτερο από αυτά αποτελεί και την τελική επιστρεφόμενη τιμή . Με τον τρόπο αυτό μειώνεται αρκετά η πιθανότητα παγίδευσης του αλγορίθμου σε κάποιο τοπικό βέλτιστο και παράλληλα αυξάνεται η πιθανότητα μεγαλύτερης κάλυψης του χώρου αναζήτησης . Το μειονέκτημα της εν λόγω προσέγγισης είναι η σχετικά αργή σε χρόνο σύγκλιση σε σχέση με την τοπολογία σμήνους πλήρους επικοινωνίας .

2.3.5 Οι παράμετροι του αλγορίθμου PSO

Οι παράμετροι του αλγορίθμου PSO φ_1 και φ_2 διαδραματίζουν ρόλο στην σύγκλισή του . Οι παράμετροι αυτοί ελέγχουν και μετριάζουν την ταχύτητα μετάβασης του μορίου προς το προσωπικό και το ολικό βέλτιστο αντίστοιχα . Οι τιμές οι οποίες θα πρέπει να δοθούν στις παραμέτρους αυτές αποτέλεσαν αντικείμενο έρευνας από τους Ozcan και Mohan (Ozcan, Mohan 1999) . Σύμφωνα με την έρευνα αυτή ο χώρος αναζήτησης περιλάμβανε μονοδιάστατο διανύσματα ενώ το σμήνος διέθετε ακριβώς 1 μόριο . Η λογική της προσέγγισης αυτής οφείλεται στο ότι οι δύο ερευνητές προσπάθησαν να επικεντρωθούν στην επίδραση των δύο παραμέτρων απομονώνοντας οποιαδήποτε άλλη παράμετρο του αλγορίθμου . Εκτός αυτού θεώρησαν ως μία την σταθερά επιτάχυνσης $\varphi = \varphi_1 + \varphi_2$, αφού όταν το σμήνος περιλαμβάνει μόνο ένα μόριο

, το προσωπικό με το ολικό βέλτιστο αναγκαστικά ταυτίζονται . Οι Ozcan και Mohan κατέληξαν στα εξής συμπεράσματα :

1. Αύξηση στην τιμή του φ έχει ως αποτέλεσμα την αύξηση της ταλάντωσης στην κίνηση του μορίου γύρω από το ολικό βέλτιστο μέχρι την τελική σύγκλισή του . Παρά το γεγονός αυτό η σύγκλιση είναι συνήθως γρηγορότερη .
2. Μείωση στην τιμή του φ έχει ως αποτέλεσμα την περισσότερο ομαλή μετάβαση του μορίου προς το τοπικό βέλτιστο αλλά έχει ως μειονέκτημα την σχετικά αργή σύγκλισή του .
3. Το άνω όριο το οποίο θα πρέπει να λαμβάνει η σταθερά επιτάχυνσης φ είναι 4 . Οι ερευνητές παρατήρησαν πειραματικά ότι τιμές μεγαλύτερες του 4 έχουν ως αποτέλεσμα τον εκφυλισμό του αλγορίθμου σε τυχαία αναζήτηση .

Όπως είναι φυσικό , οι δύο παράμετροι δεν είναι αναγκαίο να έχουν την ίδια τιμή αλλά είναι συνήθεις οι περιπτώσεις όπου ο σχεδιαστής του αλγορίθμου δίνει την ίδια τιμή και στις δύο παραμέτρους . Οι τιμές των δύο παραμέτρων εξαρτώνται από το αν ο σχεδιαστής επιθυμεί το μόριο να δίνει περισσότερη σημασία στο προσωπικό του βέλτιστο (σε αυτή την περίπτωση θα πρέπει $\varphi_1 > \varphi_2$) ή στο ολικό βέλτιστο του σμήνους (σε αυτή την περίπτωση θα πρέπει $\varphi_2 > \varphi_1$) .

Εκτός των παραμέτρων φ_1 και φ_2 σημαντικό επίσης ρόλο στην σύγκλιση του αλγορίθμου διαδραματίζει σύμφωνα με τον Eberhart (Eberhart et all 2001) και η μέγιστη τιμή της ταχύτητας v_{max} την οποία μπορεί να λάβει κάποιο μόριο του σμήνους . Η απόδοση μεγάλων τιμών στο διάνυσμα της ταχύτητας έχουν ως αποτέλεσμα την παρουσίαση μεγάλων μεταβολών στην κίνηση του μορίου , με κίνδυνο τον εκφυλισμό του αλγορίθμου σε τυχαία αναζήτηση . Αντίθετα , μικρές τιμές έχουν ως αποτέλεσμα την μικρή σε μέγεθος κίνηση των μορίων σε κάθε γενιά του αλγορίθμου και την χρονική καθυστέρηση της σύγκλισής του . Είναι προφανές ότι μεγαλύτερες τιμές του διανύσματος της ταχύτητας είναι περισσότερο επιζήμιες έναντι των μικρότερων τιμών . Για τον λόγο αυτό η ταχύτητα δεν θα πρέπει να ξεπερνά κάποιο άνω φράγμα ίσο με v_{max} . Εκτός από το άνω όριο , καλό είναι ο αλγόριθμος να μην επιτρέπει στην ταχύτητα να ξεπερνά και κάποιο κάτω όριο , ίσο με $-v_{max}$. Ειδικότερα , κατά την απόδοση νέας τιμής στην ταχύτητα v_i την χρονική στιγμή t θα πρέπει να πραγματοποιείται ο παρακάτω έλεγχος :

```
...
Αν  v_i(t) > v_max  τότε
    v_i(t) := v_max
Αλλιώς_Αν  v_i(t) < −v_max  τότε
    v_i(t) := −v_max
Τέλος_Αν
...
```

Η επιλογή της μέγιστης ταχύτητας v_{max} γίνεται συνήθως με εμπειρικό τρόπο και εξαρτάται από την φύση του προβλήματος και την δομή του χώρου αναζήτησης . Μία αρκετά αποδοτική προσέγγιση είναι αυτή όπου η μέγιστη ταχύτητα v_{max} μεταβάλλεται κατά την διάρκεια εκτέλεσης του αλγορίθμου . Ένας αρκετά συνηθισμένος τρόπος απόδοσης τιμής στην ταχύτητα v_{max} προτάθηκε από τον Abido (Abido 2001) και είναι ο εξής :

$$v_{max} = \frac{x_{max} - x_{min}}{N}$$

όπου x_{max} και x_{min} η μέγιστη και η ελάχιστη αντίστοιχα τιμή του χώρου αναζήτησης στις οποίες έχει βρεθεί κάποιο μόριο του σμήνους και N μία θετική σταθερά η οποία ορίζεται από τον σχεδιαστή του αλγορίθμου . Η εύρεση της βέλτιστης τιμής για την μέγιστη ταχύτητα v_{max} αποτελεί συνεχές αντικείμενο έρευνας .

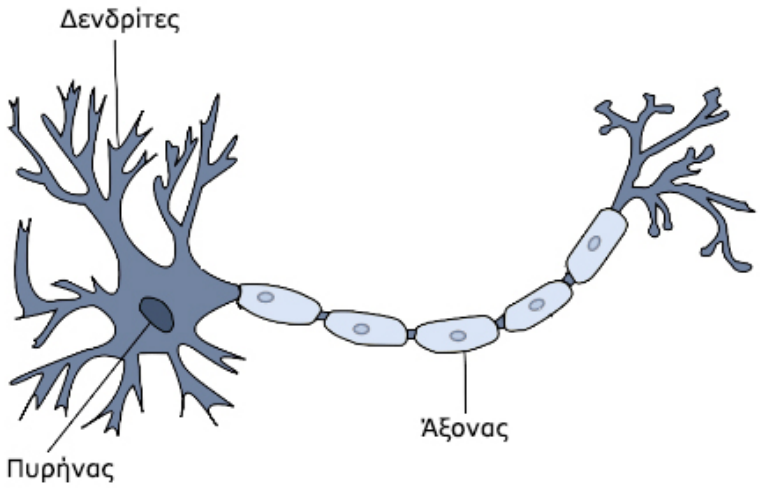
ΚΕΦΑΛΑΙΟ 3: ΤΕΧΝΗΤΑ ΝΕΥΡΩΝΙΚΑ ΔΙΚΤΥΑ ΚΑΙ ΜΕΘΟΔΟΙ ΕΚΠΑΙΔΕΥΣΗΣ

3.1 Περιγραφή των Τεχνητών Νευρωνικών Δικτύων

Τα Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ) αποτελούν έναν από τους σημαντικότερους τομείς της Μηχανικής Μάθησης . Ένα ΤΝΔ είναι ένα μαθηματικό μοντέλο το οποίο προσομοιώνει ένα βιολογικό νευρωνικό μοντέλο και το οποίο είναι σε θέση να « εκπαιδευθεί » . Με τον όρο εκπαίδευση ενός ΤΝΔ εννοούμε την ικανότητά του να προσεγγίζει μία οποιαδήποτε συνάρτηση $f(x)$, έχοντας ως δεδομένα ζεύγη τιμών $(x, f(x))$, τα οποία ονομάζονται πρότυπα εκπαίδευσης . Όσο μεγαλύτερος είναι ο αριθμός των προτύπων εκπαίδευσης τόσο καλύτερη θα είναι και η προσέγγιση της συνάρτησης από το ΤΝΔ . Αρχιτεκτονικά , το ΤΝΔ είναι ένας κατευθυνόμενος γράφος , οι κόμβοι του οποίου ονομάζονται τεχνητοί νευρώνες . Οι τεχνητοί νευρώνες προσομοιώνουν την λειτουργία των φυσικών νευρώνων του ανθρώπινου εγκεφάλου και συνολικά το ΤΝΔ προσπαθεί να προσομοιώσει την διαδικασία της ανθρώπινης μάθησης . Ο πλήρης ορισμός του ΤΝΔ καθώς και οι μέθοδοι εκπαίδευσής του περιγράφονται στην συνέχεια .

3.2 Η έννοια του Τεχνητού Νευρώνα

Ο Τεχνητός Νευρώνας (ΤΝ) αποτελεί μία προσπάθεια μοντελοποίησης ενός Βιολογικού Νευρωνικού Κυττάρου (ΒΝΚ) , το οποίο αποτελεί την κύρια συστατική μονάδα του ανθρώπινου εγκεφάλου . Ένα ΒΝΚ παρουσιάζεται στην εικόνα 8 .



Εικόνα 8 – η δομή ενός Βιολογικού Νευρωνικού Κυττάρου .

Στην εικόνα 8 μπορούμε να διακρίνουμε τα παρακάτω δομική χαρακτηριστικά τα οποία συνθέτουν ένα ΒΝΚ :

1. Οι δενδρίτες συνθέτουν τις εξωτερικές συνάψεις του ΒΝΚ . Κάθε εξωτερική σύναψη του ΒΝΚ αποτελεί το σημείο ζεύξης με άλλα ΒΝΚ . Κάθε σύναψη αποτελείται από έναν ή περισσότερους δενδρίτες . Σε κάθε δενδρίτη εισέρχεται ένα απειροελάχιστης έντασης ηλεκτρικό σήμα . Ο κάθε δενδρίτης έχει την δυνατότητα να μεταβάλλει την ένταση του ηλεκτρικού σήματος το οποίο εισέρχεται σε αυτόν , αυξάνοντάς το ή μειώνοντάς το

αντίστοιχα . Το συνολικό ηλεκτρικό σήμα το οποίο εισέρχεται στους δενδρίτες των συνάψεων εξόδου προωθείται στο κέντρο του κυττάρου , το οποίο ονομάζεται πυρήνας.

2. Ο πυρήνας του ΒΝΚ λαμβάνει το συνολικό ηλεκτρικό σήμα το οποίο διοχετεύεται σε αυτόν από τις συνάψεις του . Ο πυρήνας κάθε φορά που λαμβάνει ένα ηλεκτρικό σήμα πραγματοποιεί μία περίπλοκη βιοχημική αντίδραση . Το αποτέλεσμα της βιοχημικής αυτής αντίδρασης είναι η παραγωγή ενός νέου ηλεκτρικού σήματος , το οποίο προωθείται στο επόμενο κομμάτι του ΒΝΚ , δηλαδή τον άξονά του .
3. Ο άξονας του ΒΝΚ αποτελεί στην ουσία το σημείο εξόδου του . Το ηλεκτρικό σήμα το οποίο παράγεται από τον πυρήνα διοχετεύεται μέσω του άξονα σε άλλα ΒΝΚ του εγκεφάλου . Ο άξονας έχει και αυτός συνάψεις, αποτελούμενες από δενδρίτες , οι οποίες συνδέονται με τις συνάψεις εισόδου άλλων ΒΝΚ .

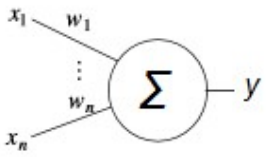
Από την παραπάνω περιγραφή του ΒΝΚ , εύκολα κάποιος μπορεί να φανταστεί ένα Βιολογικό Νευρωνικό Δίκτυο (ΒΝΔ) , το οποίο αποτελείται από ΒΝΚ . Κάθε ΒΝΚ λαμβάνει ως είσοδο ηλεκτρικά σήματα , παράγει νέα και τα αποστέλλει ως έξοδο στα επόμενα ΒΝΚ . Έτσι λοιπόν η διαδικασία της ανθρώπινης σκέψης δεν αποτελεί τίποτα περισσότερο από την μετάδοση ηλεκτρικών σημάτων μεταξύ των ΒΝΚ του ανθρώπινου εγκεφάλου .

Όσον αφορά την ταχύτητα μετάδοσης , η βιοχημική αντίδραση η οποία λαμβάνει χώρα στον πυρήνα του ΒΝΚ είναι αρκετά πιο αργή σε σχέση με τις ενέργειες που μπορεί να εκτελέσει ένας ηλεκτρονικός υπολογιστής . Το πλεονέκτημα όμως του ανθρώπινου εγκεφάλου έναντι ενός ηλεκτρονικού υπολογιστή κρύβεται σε δύο πράγματα :

1. Στην δυνατότητα της παράλληλης λειτουργίας των εκατομμυρίων ΒΝΚ που τον συνθέτουν , τα οποία δουλεύουν ταυτόχρονα .
2. Στην δυνατότητα μάθησης και εκπαίδευσής του . Ο ανθρώπινος εγκέφαλος μπορεί να εκπαιδευθεί με την χρήση παραδειγμάτων . Όσο περισσότερο ένας άνθρωπος μελετά ή ασχολείται με ένα αντικείμενο τόσο καλύτερα το μαθαίνει και τόσο μεγαλύτερες επιδόσεις θα έχει στο αντικείμενο αυτό .

Κύριος σκοπός του κλάδου της Μηχανικής Μάθησης , του οποίου τα ΤΝΔ αποτελούν κομμάτι , είναι η δημιουργία τρόπων για την εκπαίδευση υπολογιστικών συστημάτων , με την χρήση μεθόδων επιβλεπόμενης μάθησης . Με τον όρο επιβλεπόμενη μάθηση (supervised learning) αναφερόμαστε σε τεχνικές οι οποίες μπορούν να εφαρμοστούν σε συστήματα με αρχιτεκτονική εισόδου – εξόδου . Αυτό το οποίο είναι πρακτικά επιθυμητό είναι δεδομένων κάποιων παραδειγμάτων της μορφής (είσοδος , έξοδος) , τα οποία έχουν κοινή θεματολογία , το υπολογιστικό σύστημα να είναι σε θέση να δίνει την σωστή έξοδο για άγνωστες για αυτό εισόδους , παρόμοιας θεματολογίας . Στην ουσία , όπως αναφέρθηκε παραπάνω , αυτό το οποίο είναι επιθυμητό είναι η προσέγγιση μιας οποιαδήποτε συνάρτησης $f(x)$, έχοντας ως δεδομένα ζεύγη τιμών $(x, f(x))$.

Η αρχιτεκτονική των ΒΝΚ και των ΒΝΔ κίνησε το ενδιαφέρον των ερευνητών με αποτέλεσμα την προσπάθεια προσομοίωσης των ΒΝΔ . Αρχικό σημείο στην όλη προσπάθεια αποτελεί η μοντελοποίηση ενός ΒΝΚ με κάποιο τεχνητό αντίστοιχο πρότυπο , το οποίο ονομάζεται Τεχνητός Νευρώνας (ΤΝ) . Πολύ συχνή ονομασία για έναν ΤΝ αποτελεί επίσης και αυτή του αισθητήρα (perception) (Rojas 1996) . Ένας αισθητήρας παρουσιάζεται στην εικόνα 9 .



Εικόνα 9 – παράδειγμα απλού αισθητήρα .

Ένας αισθητήρας αποτελεί ένα μαθηματικό μοντέλο το οποίο μπορεί να προσομοιωθεί από ένα υπολογιστικό σύστημα . Ένας αισθητήρας έχει ένα σύνολο από πραγματικές εισόδους $\{x_1, x_2, \dots, x_n\}$ με $x_1, x_2, \dots, x_n \in \mathbb{R}$. Εκτός αυτού περιλαμβάνει και ένα σύνολο με πραγματικά βάρη $\{w_1, w_2, \dots, w_n\}$ με $w_1, w_2, \dots, w_n \in \mathbb{R}$. Σε κάθε είσοδο αντιστοιχεί κι ένα βάρος . Όταν όλες οι πραγματικές είσοδοι του αισθητήρα λάβουν τιμές, τότε ο αισθητήρας ενεργοποιείται . Ειδικότερα

, ο αισθητήρας δημιουργεί ένα άθροισμα, του οποίου οι όροι είναι οι τιμές των εισόδων πολλαπλασιασμένες με τις τιμές των αντίστοιχων βαρών . Το άθροισμα αυτό ονομάζεται ενεργοποίηση (activation) του αισθητήρα και υπολογίζεται ως εξής :

$$\sum_{i=1}^{n} x_i \cdot w_i$$

Μετά τον υπολογισμό του αθροίσματος , ο αισθητήρας υπολογίζει την έξοδό του y . Κάθε αισθητήρας διαθέτει μία συνάρτηση $f: \mathbb{R} \to \mathbb{R}$, η οποία ονομάζεται συνάρτηση ενεργοποίησης (activation function) . Το αποτέλεσμα της συνάρτησης ενεργοποίησης για είσοδο ίση με την ενεργοποίηση του αισθητήρα αποτελεί και την έξοδό του y . Έτσι λοιπόν :

$$y = f\left(\sum_{i=1}^{n} x_i \cdot w_i\right)$$

Η πλέον συνηθέστερη συνάρτηση ενεργοποίησης είναι η βηματοειδής συνάρτηση (step function) , η οποία ορίζεται ως εξής :

$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = \begin{cases} 1, & \sum_{i=1}^{n} w_i \cdot x_i \geq 0 \\ 0, & \sum_{i=1}^{n} w_i \cdot x_i < 0 \end{cases}$$

Πολλές φορές είναι επιθυμητό η βηματοειδής συνάρτηση να επιστρέφει 1 ή 0 αντίστοιχα ανάλογα με το αν η ενεργοποίηση είναι μεγαλύτερη ή ίση ενός κατωφλίου (threshold) , το οποίο συμβολίζεται με θ . Έτσι λοιπόν , η μορφή αυτή της βηματοειδούς συνάρτησης είναι η εξής :

$$y = f\left(\sum_{i=1}^{n} w_i \cdot x_i\right) = \begin{cases} 1, & \sum_{i=1}^{n} w_i \cdot x_i \geq \theta \\ 0, & \sum_{i=1}^{n} w_i \cdot x_i < \theta \end{cases}$$

Επειδή η παραπάνω έκδοση της βηματοειδούς συνάρτησης είναι ιδιαίτερα δημοφιλής , μπορούμε να προσθέσουμε το κατώφλι θ σαν βάρος στον αισθητήρα . Η είσοδος στην οποία το βάρος θα αντιστοιχίζεται θα είναι πάντοτε σταθερή και ίση με -1 . Αυτό θα έχει ως αποτέλεσμα η ενεργοποίηση να ορίζεται πλέον ως εξής :

$$\sum_{i=1}^{n} x_i \cdot w_i - \theta$$

και εύκολα πλέον μπορούμε να δείξουμε ότι :

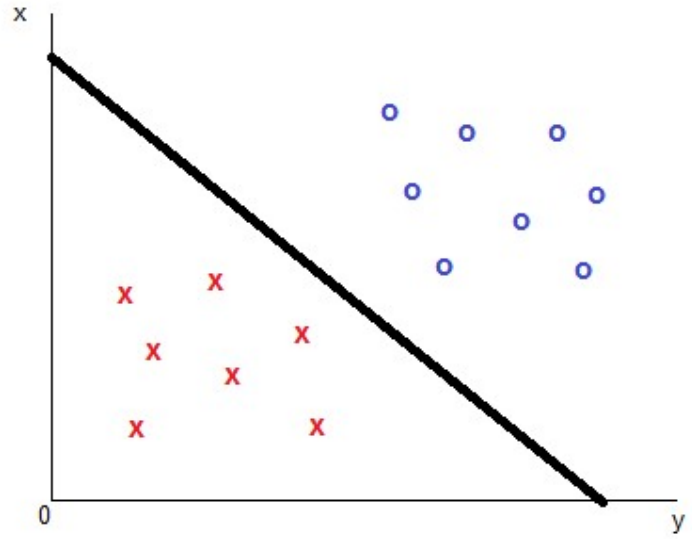
$$\sum_{i=1}^{n} w_i \cdot x_i - \theta \geq 0 \Rightarrow \sum_{i=1}^{n} w_i \cdot x_i \geq \theta$$

και

$$\sum_{i=1}^{n} w_i \cdot x_i - \theta < 0 \Rightarrow \sum_{i=1}^{n} w_i \cdot x_i < \theta$$

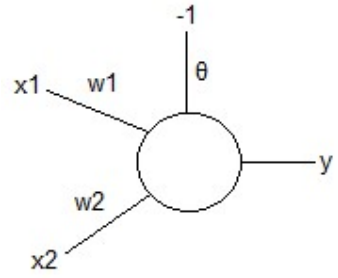
3.3 Χρησιμότητα του απλού αισθητήρα

Η χρησιμότητα του απλού αισθητήρα είναι ότι μπορεί εύκολα να επιλύσει προβλήματα γραμμικής διαχωρισιμότητας . Ένα πρόβλημα γραμμικής διαχωρισιμότητας είναι στην ουσία ένα πρόβλημα διαχωρισμού δεδομένων με μία ευθεία γραμμή . Ας υποθέσουμε ότι έχουμε ένα σύνολο στοιχείων $\{k_1, k_2, \dots, k_n\}$ όπου το κάθε ένα από αυτά ανήκει σε δύο δυνατές κλάσεις A ή B . Αν τα στοιχεία αυτά μπορούν να αναπαρασταθούν γραφικά στο 2-διάστατο επίπεδο και οι δύο κλάσεις μπορούν να διαχωριστούν με μία ευθεία γραμμή , τότε το πρόβλημα είναι γραμμικώς διαχωρίσιμο . Ένα γραμμικώς διαχωρίσιμο πρόβλημα παρουσιάζεται στην εικόνα 10 .



Εικόνα 10 – παράδειγμα γραμμικώς διαχωρίσιμου προβλήματος .

Ο απλός αισθητήρας έχει την δυνατότητα επίλυσης ενός γραμμικώς διαχωρίσιμου προβλήματος . Ας θεωρήσουμε έναν απλό αισθητήρα με δύο μεταβλητές εισόδους , σαν κι αυτόν ο οποίος εμφανίζεται στην εικόνα 11 .



Εικόνα 11 – ένας αισθητήρας δύο εισόδων .

Η ενεργοποίηση του παραπάνω αισθητήρα θα είναι ίση με :

$$w_1 x_1 + w_2 x_2 - \theta$$

Αν εξισώσουμε την ενεργοποίηση με το μηδέν, τότε προκύπτει ότι :

$$w_1 x_1 + w_2 x_2 - \theta = 0$$

το οποίο αναπαριστά μία ευθεία στο σύστημα αξόνων με άξονες τους x_1, x_2, αφού οι εξισώσεις ευθειών είναι της παρακάτω μορφής :

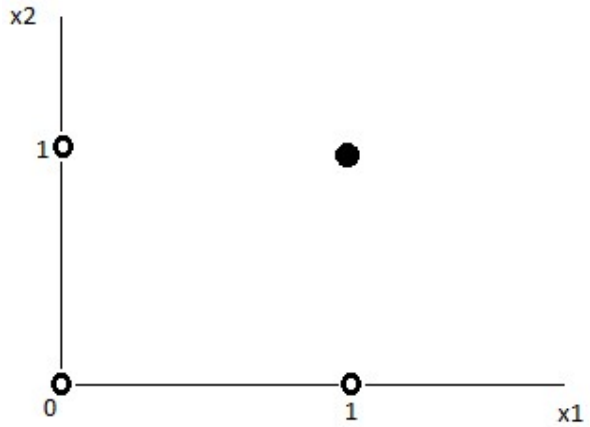
$$a x_1 + b x_2 + c = 0$$

Αν χρησιμοποιήσουμε ως συνάρτηση ενεργοποίησης του αισθητήρα την βηματοειδή , τότε τα σημεία τα οποία βρίσκονται γεωμετρικά « πάνω » από την ευθεία , δηλαδή αυτά για τα οποία ισχύει ότι $w_1 x_1 + w_2 x_2 - \theta > 0$, λαμβάνουν την τιμή 1 από την συνάρτηση ενεργοποίησης . Αντίστοιχα τα σημεία τα οποία βρίσκονται γεωμετρικά « κάτω » από την ευθεία , δηλαδή αυτά για τα οποία ισχύει ότι $w_1 x_1 + w_2 x_2 - \theta < 0$, λαμβάνουν την τιμή 0 . Αν ένα σημείο είναι ακριβώς πάνω στην ευθεία , τότε ισχύει ότι $w_1 x_1 + w_2 x_2 - \theta = 0$ και συνεπώς δεν μπορεί να κατηγοριοποιηθεί σε καμία από τις δύο κατηγορίες . Όπως είναι λοιπόν προφανές , ο απλός αισθητήρας είναι σε θέση να λειτουργήσει ως γραμμικός διαχωριστής και η εξίσωση της ευθείας που του αντιστοιχεί ονομάζεται γραμμή απόφασης .

Στο σημείο αυτό κρίνεται σκόπιμο να δώσουμε ένα παράδειγμα γραμμικού διαχωρισμού . Ας θεωρήσουμε λοιπόν τη γνωστή πύλη AND 2 εισόδων , της οποίας ο πίνακας αληθείας είναι ο παρακάτω :

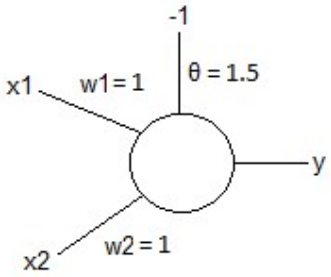
x_1	x_2	y
0	0	0
0	1	0
1	0	0
1	1	1

Αν αναπαραστήσουμε τον πίνακα αληθείας ως σημεία δύο διαστάσεων στο 2-διάστατο επίπεδο , τότε έχουμε το σχήμα που παρουσιάζεται στην εικόνα 11 .



Εικόνα 11 – ο πίνακας αληθείας της πύλης AND ως σημεία στο 2-διάστατο επίπεδο .

Στην παραπάνω εικόνα κάθε είσοδος έχει αναπαρασταθεί ως ένα 2-διάστατο σημείο , με συντεταγμένες (0,0), (0,1), (1,0) και (1,1) αντίστοιχα . Στα σημεία όπου η πύλη AND έχει ως έξοδο την τιμή 0 εμφανίζονται άσπρα ενώ το σημείο όπου η πύλη AND έχει την τιμή 1 εμφανίζεται μαύρο αντίστοιχα . Ας θεωρήσουμε τώρα τον παρακάτω αισθητήρα 2 εισόδων που παρουσιάζεται στην εικόνα 12 .

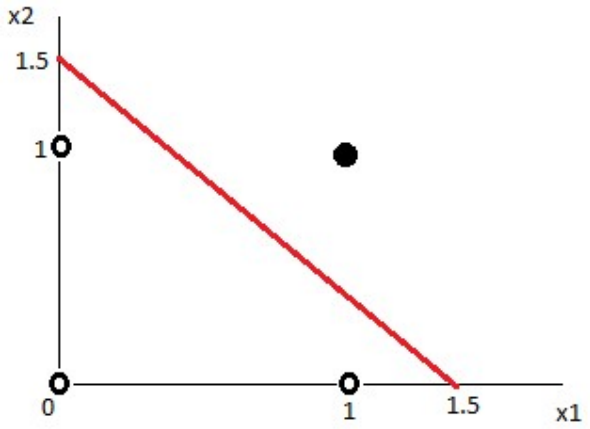


Εικόνα 12 – αισθητήρας ο οποίος λειτουργεί ως πύλη AND .

Ο παραπάνω αισθητήρας αντιστοιχείς την εξής γραμμή απόφασης :

$$x_1 + x_2 - 1.5 = 0$$

η οποία γραφικά αναπαρίσταται στην εικόνα 13 .



Εικόνα 13 – η γραμμή απόφασης του παραπάνω αισθητήρα .

Η γραμμή αυτή απόφασης διαχωρίζεται τις δύο κλάσεις σημείων (άσπρα και μαύρα) . Πράγματι , ο αισθητήρας λειτουργεί ως πύλη AND , αν έχει ως συνάρτηση ενεργοποίησης την βηματοειδή συνάρτηση , αφού :

x_1	x_2	θ	ενεργοποίηση	y
0	0	1.5	-1.5 < 0	0
0	1	1.5	-0.5 < 0	0
1	0	1.5	-0.5 < 0	0
1	1	1.5	0.5 > 0	1

Στην περίπτωση που οι μεταβλητές είσοδοι του αισθητήρα ήταν περισσότερες από 2 , η γραμμή απόφασης του αισθητήρα θα αντιστοιχούσε σε ένα υπερεπίπεδο διάστασης ίσης με τον αριθμό των εισόδων . Και στην περίπτωση αυτή , για να μπορεί ο αισθητήρας να εφαρμοστεί για την επίλυση του προβλήματος , θα πρέπει τα σημεία εισόδου να μπορούν να διαχωριστούν από το υπερεπίπεδο το οποίο σχηματίζεται .

Ανεξαρτήτως της διάστασης των σημείων εισόδου , η χρήση ενός αισθητήρα για την επίλυση ενός γραμμικά διαχωρίσιμου προβλήματος εξαρτάται από τις τιμές που έχουν τα βάρη του . Η απόδοση των σωστών τιμών στα βάρη ενός αισθητήρα , δεδομένων των σημείων ενός γραμμικά διαχωρίσιμου προβλήματος και της κλάσης του καθενός από αυτά, ονομάζεται εκπαίδευση (training) . Είναι πολύ σύνηθες επίσης τα σημεία εισόδου , με βάση τα οποία ο αισθητήρας εκπαιδεύεται, να ονομάζεται παραδείγματα εκπαίδευσης. Ένας απλός αλγόριθμος εκπαίδευσης αισθητήρων , ανεξαρτήτως διάστασης των σημείων εισόδου , παρουσιάζεται παρακάτω . Ο αλγόριθμος αυτός ονομάζεται Κανόνας Δέλτα (Rosenblatt 1958) και είναι ο πλέον δημοφιλέστερος για την εκπαίδευση του απλού αισθητήρα .

// Είσοδος

- τα γραμμικώς διαχωρίσιμα παραδείγματα εκπαίδευσης $T = \{x_1, x_2, \dots, x_n\}$, διάστασης p.
- μία συνάρτηση $d: T \to \{0,1\}$, η οποία αντιστοιχίζει το κάθε παράδειγμα εκπαίδευσης σε μία από τις δύο κλάσεις, οι οποίες αναπαρίστανται με 0 και 1.
- ο αισθητήρας με n στον αριθμό μεταβλητά βάρη $\{w_1, w_2, \dots, w_n\}$ και κατώφλι θ.
- η βηματοειδής συνάρτησης $f: \mathbb{R} \to \{0,1\}$ του αισθητήρα.
- μία πραγματική τιμή a (η οποία ονομάζεται ρυθμός μάθησης)

// Επεξεργασία

1. Αρχικοποίηση με τυχαίες τιμές τα μεταβλητά βάρη $\{w_1, w_2, \dots, w_n\}$ του αισθητήρα, καθώς και το κατώφλι θ.

2. $Total\_Error := 0$
3. Για κάθε παράδειγμα εκπαίδευσης $x_i \in T$ επανάλαβε
   3.1. Υπολόγισε την ενεργοποίηση U_i του αισθητήρα $U_i := \sum_{j=1}^{k} x_{i,j} \cdot w_i - \theta$.
   3.2. Υπολόγισε την έξοδο του αισθητήρα $y_i = f(U_i)$.
   3.3. $Total\_Error := Total\_Error + |d_i - y_i|$.
   3.4. Αν $y_i - d_i \neq 0$ τότε
      3.4.1. Ανανέωσε κάθε βάρος του αισθητήρα. Για κάθε w_i επανάλαβε
         3.4.1.1. $w_i' = w_i + a \cdot (d_i - y_i)$
      3.4.2. Τέλος_Επανάληψης
      3.4.3. Ανανέωσε την τιμή του κατωφλίου $\theta' = \theta + a \cdot (d_i - y_i)$
   3.5. Τέλος_Αν
4. Τέλος_Επανάληψης
5. Αν $Total\_Error \neq 0$ τότε
   5.1. Πήγαινε στο βήμα 2 του αλγορίθμου
6. Τέλος_Αν

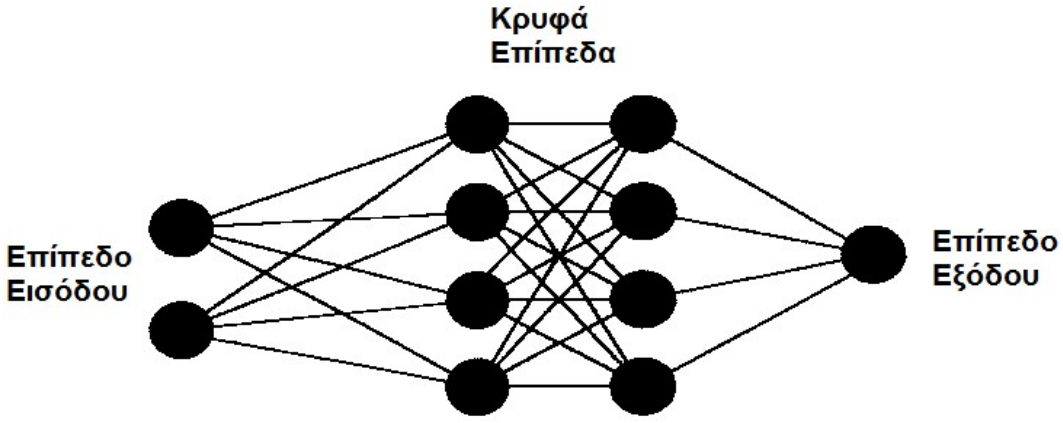
// Έξοδος

- ο αισθητήρας με τις τιμές των βαρών (μεταβλητών και κατωφλίου) μετά το τέλος της εκπαίδευσης.

Ο παραπάνω αλγόριθμος μπορεί να εφαρμοστεί και σε ΤΝ οι οποίοι δεν έχουν απαραίτητα ως συνάρτηση ενεργοποίησης την βηματοειδή . Ο αλγόριθμος συγκλίνει και τερματίζει αν τα παραδείγματα εισόδου είναι όντως διαχωρίσιμα και μπορούν να διαχωριστούν από ένα υπερεπίπεδο στον χώρο διάστασης n . Αν τα παραδείγματα εισόδου δεν είναι διαχωρίσιμα , τότε ο αλγόριθμος συνεχίζει την εκτέλεσή του επ' άπειρον . Κριτήριο τερματισμού του Κανόνα Δέλτα είναι ο αισθητήρας να απαντά σωστά (δηλαδή να ταξινομεί στην σωστή κλάση) για κάθε παράδειγμα εισόδου .

3.4 Τα Τεχνητά Νευρωνικά Δίκτυα

Η σύνδεση των εξόδων κάποιων ΤΝ με τις εισόδους κάποιων άλλων έχει ως αποτέλεσμα την κατασκευή μιας δομής η οποία ονομάζεται Τεχνητό Νευρωνικό Δίκτυο (ΤΝΔ) . Τα ΤΝΔ χωρίζονται σε υποκατηγορίες ανάλογα με την κατανομή των ΤΝ που το αποτελούν στον χώρο αλλά και την φιλοσοφία σύνδεσής τους . Η πλέον συνηθέστερη αρχιτεκτονική των ΤΝΔ , είναι τα ΤΝΔ εμπρόσθιας τροφοδότησης (feed-forward artificial neural networks) . Ένα ΤΝΔ εμπρόσθιας τροφοδότησης παρουσιάζεται παρακάτω , στην εικόνα 14 .



Εικόνα 14 – η αρχιτεκτονική δομή ενός ΤΝΔ εμπρόσθιας τροφοδότησης .

Από την παραπάνω εικόνα μπορούμε να δούμε ότι ένα ΤΝΔ εμπρόσθιας τροφοδότησης χωρίζεται σε 3 νοητά επίπεδα :

- 1 επίπεδο εισόδου
- 1 ή περισσότερα κρυφά επίπεδα
- 1 επίπεδο εξόδου

Το επίπεδο εισόδου αποτελείται από έναν ή περισσότερους νευρώνες εισόδου . Οι νευρώνες εισόδου είναι ΤΝ οι οποίοι έχουν διαφορετική λειτουργία από τους υπόλοιπους . Ειδικότερα , οι συγκεκριμένοι νευρώνες απλά λαμβάνουν μία είσοδο και την αναμεταδίδουν , χωρίς να προχωρούν σε κάποια περαιτέρω επεξεργασία . Αυτό πρακτικά σημαίνει ότι δεν υπολογίζουν κάποια ενεργοποίηση για την είσοδο αυτοί και επίσης δεν είναι εφοδιασμένοι με κάποια συνάρτηση ενεργοποίησης .

Τα κρυφά επίπεδα αποτελούνται από έναν ή περισσότερους ΤΝ . Τα κρυφά επίπεδα αναλαμβάνουν την επεξεργασία των εισόδων του ΤΝΔ . Επειδή το ΤΝΔ είναι εμπρόσθιας τροφοδότησης , οι ΤΝ ενός επιπέδου τροφοδοτούν την έξοδό τους μόνο σε νευρώνες επόμενων επιπέδων. Το πλέον συνηθέστερο είναι οι ΤΝ ενός επιπέδου να τροφοδοτούν τους νευρώνες μόνο του ακριβώς επόμενου επιπέδου , χωρίς αυτό φυσικά να είναι απαραίτητο . Ο αριθμός των κρυφών επιπέδων ενός ΤΝΔ μπορεί επίσης να ποικίλει . Σε γενικές γραμμές δεν υπάρχει συγκεκριμένος τρόπος εύρεσης του αριθμού κρυφών επιπέδων που απαιτούνται για την κατασκευή του βέλτιστου ΤΝΔ που επιλύει ένα πρόβλημα. Αυτό που έχει όμως αποδειχθεί και ονομάζεται Καθολικό Θεώρημα Προσέγγισης (Universal approximation theorem) από τον Cybenko (Cybenko 1989) είναι ότι κάθε συνάρτηση $f: \mathbb{R}^n \to \mathbb{R}^k$ μπορεί να προσεγγιστεί , σε οποιαδήποτε ακρίβεια , από ένα ΤΝΔ εμπρόσθιας τροφοδότησης με ένα ακριβώς κρυφό επίπεδο όπου οι ΤΝ έχουν ως συνάρτηση ενεργοποίησης την σιγμοειδή . Η σιγμοειδής συνάρτηση $S(u)$ ορίζεται ως εξής :

$$S(u) = \frac{1}{1 + e^{-cu}}$$

όπου u η ενεργοποίηση του ΤΝ και c μια πραγματική σταθερά . Το Καθολικό Θεώρημα Προσέγγισης αποδεικνύει απλώς την ύπαρξη ενός ΤΝΔ για κάθε πρόβλημα για το οποίο υπάρχει κάποια συνάρτηση που να το περιγράφει αλλά δεν μας δίνει καμία πληροφορία για τον αριθμό των ΤΝ του κρυφού επιπέδου αλλά ούτε και για τις τιμές των βαρών .

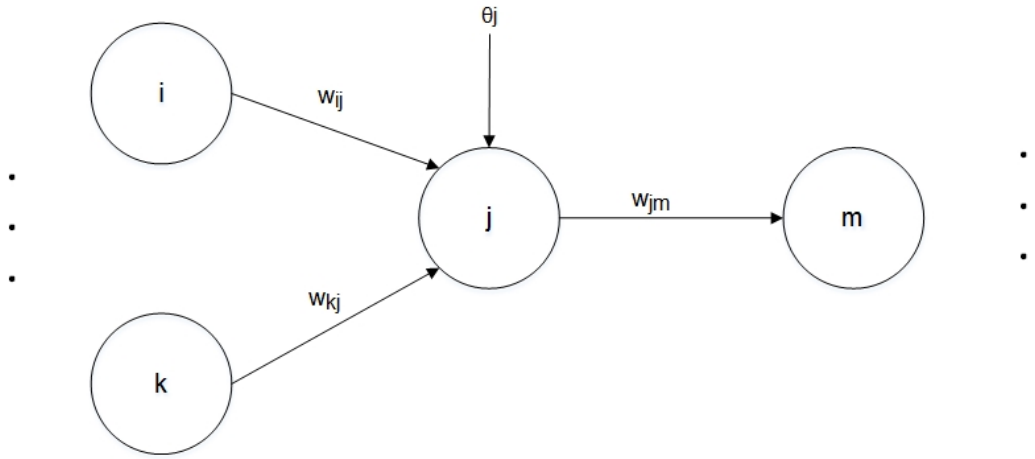
Το επίπεδο εξόδου αποτελείται από έναν ή περισσότερους ΤΝ , οι οποίοι δεν διαφέρουν σε τίποτα από τους ΤΝ των κρυφών επιπέδων . Ο μόνος λόγος διαχωρισμού του επιπέδου εξόδου από τα κρυφά επίπεδα είναι επειδή είναι το τελευταίο στην σειρά και αποτελεί το σημείο από το οποίο ο σχεδιαστής του ΤΝ θα συλλέξει τα δεδομένα εξόδου .

3.5 Εκπαίδευση ενός Τεχνητού Νευρωνικού Δικτύου

Με τον όρο εκπαίδευση ενός ΤΝΔ εννοούμε την απόδοση τιμών στα βάρη των ΤΝ που το απαρτίζουν , έτσι ώστε για συγκεκριμένες τιμές στις εισόδους το ΤΝΔ να απαντά με αντίστοιχες τιμές στις εξόδους του . Μπορούμε να ορίσουμε την εκπαίδευση ενός ΤΝΔ με τυπικό τρόπο παρακάτω .

Έστω ένα ΤΝΔ το οποίο αποτελείται από ένα επίπεδο εισόδου , ένα ή περισσότερα κρυφά επίπεδα και ένα επίπεδο εξόδου . Το ΤΝΔ είναι εμπρόσθιας τροφοδότησης και κάθε επίπεδο μεταδίδει πληροφορία προς τα επόμενα . Το επίπεδο εισόδου αποτελείται από νευρώνες εισόδου , οι οποίοι απλά μεταδίδουν προς τα εμπρός την είσοδό τους ενώ τα κρυφά επίπεδα και το επίπεδο εξόδου από ΤΝ οι οποίοι λειτουργούν με τον τρόπο με τον οποίο περιγράφηκε προηγουμένως . Για να τους διαχωρίσουμε από τους νευρώνες εισόδου , θα τους ονομάσουμε υπολογιστικούς νευρώνες . Κάθε ΤΝ (εισόδου ή υπολογιστικός) χαρακτηρίζεται από έναν μοναδικό θετικό ακέραιο αριθμό . Κάθε νευρώνας εισόδου έχει ακριβώς μία είσοδο την οποία μεταδίδει στους υπολογιστικούς νευρώνες των επόμενων επιπέδων . Κάθε υπολογιστικός νευρώνας έχει ένα σύνολο από μία ή περισσότερες συνάψεις εισόδου και μία ακριβώς έξοδο , την οποία μπορεί να διαμοιράσει σε υπολογιστικούς νευρώνες επόμενων επιπέδων . Κάθε

εισερχόμενη σύναψη του νευρώνα j χαρακτηρίζεται από ένα βάρος w_{ij} όπου i είναι ο ΤΝ από τον οποίον η σύναψη λαμβάνει είσοδο (ΤΝ προηγούμενου επιπέδου) . Μοναδική εξαίρεση αποτελεί η σύναψη η οποία έχει πάντοτε σταθερή είσοδο , της οποίας το βάρος (κατώφλι) θα συμβολίζεται με θ_j . Ένα παράδειγμα της παραπάνω περιγραφής παρουσιάζεται στην Εικόνα 15 .



Εικόνα 15 – παράδειγμα ονομασίας των βαρών του νευρώνα j .

Έστω επίσης ένα σύνολο διανυσμάτων εισόδου $T = \{x_1, x_2, \dots, x_n\}$ διάστασης p , όπου το p είναι ίσο με τον αριθμό των νευρώνων εισόδου , έτσι ώστε κάθε στοιχείο του διανύσματος x_i να αντιστοιχεί σε έναν νευρώνα εισόδου . Έστω επίσης ένα σύνολο διανυσμάτων εξόδου $D = \{d_1, d_2, \dots, d_n\}$ διάστασης q , όπου το q είναι ίσο με τον αριθμό των νευρώνων εξόδου , έτσι ώστε κάθε στοιχείο του διανύσματος d_i να αντιστοιχεί σε έναν νευρώνα εξόδου . Ορίζουμε ως εκπαίδευση ενός ΤΝΔ οποιαδήποτε διαδικασία ακολουθείται για την εύρεση των τιμών των βαρών w_{ij} και θ_j κάθε υπολογιστικού νευρώνα j , έτσι ώστε για κάθε διάνυσμα εισόδου x_i το ΤΝΔ να εμφανίζει στην έξοδό του το αντίστοιχο διάνυσμα εξόδου d_i . Όπως εύκολα μπορεί κάποιος να δει στο σημείο αυτό , η εκπαίδευση ενός ΤΝΔ πρόκειται για ένα πρόβλημα αναζήτησης , σε έναν χώρο με αρκετά μεγάλο αριθμό πιθανών λύσεων . Για την εκπαίδευση των ΤΝΔ έχει αναπτυχθεί ένας αρκετά μεγάλος αριθμός αλγορίθμων , με πλέον δημοφιλέστερο αυτόν της οπισθοδιάδοσης του λάθους , τον οποίον θα αναλύσουμε παρακάτω .

Η χρησιμότητα της εκπαίδευσης του ΤΝΔ έγκειται στο γεγονός του ότι ένα ΤΝΔ μπορεί να « μάθε ι» από ήδη υπάρχοντα παραδείγματα και να γενικεύσει αυτά τα οποία έμαθε σε άλλα . Για παράδειγμα έστω ότι τα διανύσματα εισόδου αποτελούν μετρήσεις μετεωρολογικών μεγεθών μιας συγκεκριμένης περιοχής σε διαδοχικές ημέρες ενώ τα διανύσματα εξόδου αντιστοιχούν στην τιμή της θερμοκρασίας της περιοχής αυτής τις αντίστοιχες ημέρες . Με την υπόθεση του ότι παρόμοιες μετρήσεις με αυτές των διανυσμάτων εισόδου έχουν ως αποτέλεσμα παρόμοιες θερμοκρασίες με αυτές των διανυσμάτων εξόδου , εκπαιδεύοντας το ΤΝΔ με τα ήδη υπάρχοντα δεδομένα ευελπιστούμε ότι το ΤΝΔ θα είναι σε θέση να κάνει σωστές προβλέψεις θερμοκρασίες και για νέες μελλοντικές μετρήσεις που θα παρουσιαστούν σε αυτό μετέπειτα . Όσο περισσότερο η λειτουργία ενός συστήματος μπορεί να αναπαρασταθεί με την χρήση μιας συνάρτησης, τόσο καλύτερα αυτό μπορεί να προσεγγιστεί από ένα ΤΝΔ , αφού σύμφωνα με το Καθολικό Θεώρημα Προσέγγισης που περιγράψαμε παραπάνω , τα ΤΝΔ μπορούν θεωρητικά να προσεγγίσουν οποιαδήποτε συνάρτηση με οποιαδήποτε ακρίβεια . Ένα πρόβλημα στην εκπαίδευση των ΤΝΔ αποτελεί η εύρεση της κατάλληλη αρχιτεκτονικής την οποία αυτό θα πρέπει να έχει , δηλαδή τον αριθμό των κρυφών επιπέδων και τον αριθμό των νευρώνων σε κάθε ένα από αυτά . Στην περίπτωση που επιθυμούμε και την εύρεση της κατάλληλης αρχιτεκτονικής , τότε ο χώρος αναζήτησης του προβλήματος αυξάνει κατά πολύ σε μέγεθος , αφού πλέον από πρόβλημα εύρεσης των βαρών ενός συγκεκριμένου ΤΝΔ μετατρέπεται σε πρόβλημα εύρεσης των βαρών διαφορετικών ΤΝΔ .

3.6 Ο αλγόριθμος Οπισθοδιάδοσης του Λάθους

Όπως αναφέρθηκε παραπάνω , ο περισσότερο διαδεδομένος αλγόριθμος εκπαίδευσης ενός ΤΝΔ είναι ο αλγόριθμος Οπισθοδιάδοσης του Λάθους (Back Propagation – BP) . Ο αλγόριθμος αυτός προσπαθεί να ελαχιστοποιήσει το τετραγωνικό σφάλμα της τρέχουσας εξόδου του ΤΝΔ σε σχέση με την επιθυμητή έξοδο . Ειδικότερα , αν έχουμε n στον αριθμό παραδείγματα εκπαίδευσης , με d_i την επιθυμητή έξοδο του δικτύου για το στοιχείο $x_i \in T$ και με y_i την πραγματική έξοδο του δικτύου για το στοιχείο αυτό , τότε ο αλγόριθμος BP προσπαθεί να ελαχιστοποιήσει το παρακάτω μέγεθος :

$$Total\_Error = \sum_{i=1}^{n} (d_i - y_i)^2$$

Αν το επίπεδο εξόδου περιλαμβάνει παραπάνω του ενός νευρώνες εξόδου , τότε η διαφορά της επιθυμητής με την πραγματική έξοδο πρόκειται στην ουσία για μία αφαίρεση διανυσμάτων , η οποία ορίζεται ως εξής (υποθέτουμε ότι τα διανύσματα είναι διάστασης q) :

$$d_i - y_i = \sum_{j=1}^{q} \left(d_{i,j} - y_{i,j} \right)^2$$

όπου $d_{i,j}$ είναι το στοιχείο j του διανύσματος d_i (αντίστοιχα και για το δεύτερο διάνυσμα) .

Η διαδικασία εκπαίδευσης του ΤΝΔ γίνεται σε κύκλους εκπαίδευσης , όπου σε κάθε κύκλο πραγματοποιούνται ακριβώς οι ίδιες ενέργειες . Η διαδικασία εκπαίδευσης σταματά μέχρι να ικανοποιηθεί κάποιο κριτήριο τερματισμού , το οποίο συνήθως είναι είτε η ολοκλήρωση ενός αριθμού κύκλων είτε ότι το συνολικό τετραγωνικό σφάλμα των εξόδων μειωθεί κάτω από ένα κατώφλι ε ($Total\_Error < \varepsilon$) . Οι ενέργειες οι οποίες πραγματοποιούνται σε έναν κύκλο εκπαίδευσης πραγματοποιούνται διαδοχικά για κάθε στοιχείο εκπαίδευσης του συνόλου T. Ειδικότερα , οι υπολογισμοί για ένα στοιχείο $x_i \in T$ σε κάποιον κύκλο εκπαίδευσης είναι οι παρακάτω :

1. *Εμπρόσθιο πέρασμα :* στο στάδιο αυτό υπολογίζεται η έξοδος του κάθε νευρώνα σε κάθε επίπεδο του ΤΝΔ, με απώτερο στόχο τον υπολογισμό των εξόδων του τελευταίου επιπέδου . Για κάθε νευρώνα j , με n εισόδους , η έξοδος υπολογίζεται ως εξής :

   $$u_j = \sum_{i=1}^{n} w_i \cdot x_i - \theta_j \; και \; y_j = f\left(u_j\right)$$

   Στην περίπτωση που πρόκειται για νευρώνα κρυφού επίπεδου , η κάθε είσοδος x_i αποτελεί έξοδο νευρώνα προηγούμενου επιπέδου ενώ αν πρόκειται για νευρώνα του πρώτου επιπέδου (επιπέδου εισόδου) η έξοδός του ισούται με την είσοδό του .
2. *Υπολογισμός σφαλμάτων :* στο στάδιο αυτό υπολογίζεται το σφάλμα για κάθε νευρώνα εξόδου j . Το σφάλμα υπολογίζεται ως εξής :

   $$e_j = d_j - y_j$$
3. *Πέρασμα προς τα πίσω :* στο στάδιο αυτό το σφάλμα των νευρώνων εξόδου διαδίδεται προς τα πίσω , στους νευρώνες των προηγουμένων επιπέδων . Πιο συγκεκριμένα , για κάθε νευρώνα του ΤΝΔ υπολογίζεται το παρακάτω μέγεθος , το οποίο στην ουσία αναπαριστά τον « βαθμό ευθύνης » που έχει ο νευρώνας :
   - $\delta_j = f^{'}(u_j) \cdot e_j$, αν ο j είναι νευρώνας εξόδου
   - $\delta_j = f^{'}(u_j) \cdot \sum_k w_{jk} \cdot \delta_k$, αν ο j είναι κρυφός νευρώνας και k ο κάθε νευρώνας στον οποίον ο j δίνει έξοδο .
4. *Υπολογισμός των μεταβολών των βαρών :* για κάθε μεταβλητό βάρος w_{ij} του νευρώνα j , το οποίο λαμβάνει ως είσοδο την έξοδο του νευρώνα i , αλλά και για το κατώφλι θ_j , υπολογίζεται η μεταβολή του ως εξής :

$$\Delta w_{ij} = a \cdot \delta_j \cdot y_i$$

$$\theta_j = a \cdot \delta_j \cdot (-1)$$

όπου a μία πραγματική σταθερά η οποία ονομάζεται ρυθμός μάθησης (learning rate) .

5. *Ανανέωση των βαρών:* κάθε βάρος w_{ii} , το οποίο επισυνάπτεται στην i-οστή είσοδο του νευρώνα j ανανεώνει την τιμή του ως εξής :

$$w_{ij}^{'} = w_{ij} + \Delta w_{ij}$$

Εκτός από τα μεταβλητά βάρη, ανανεώνονται επίσης και τα κατώφλια :

$$\theta_j^{'} = \theta_j + \Delta\theta_j$$

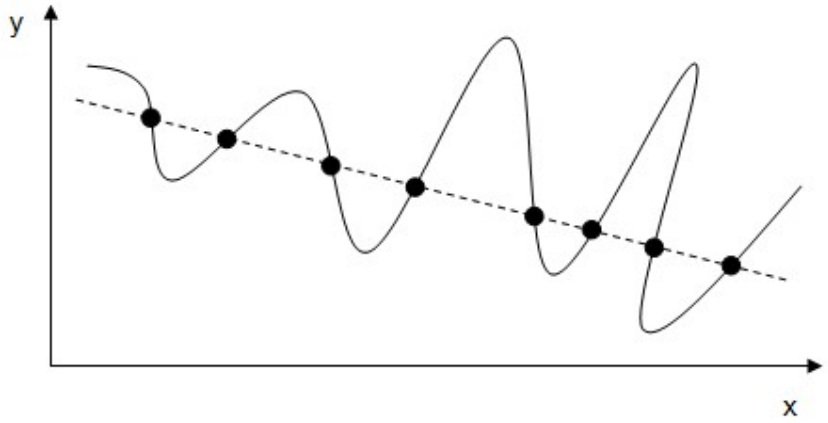
Με την πάροδο ενός αριθμού κύκλων εκπαίδευσης , το συνολικό τετραγωνικό σφάλμα μειώνεται και με τον τρόπο αυτό το ΤΝΔ εκπαιδεύεται . Σημαντικό ρόλο στην χρονική επίδοση του αλγορίθμου διαδραματίζει ο ρυθμός μάθησης a . Μεγάλες τιμές στην παράμετρο a έχουν ως αποτέλεσμα την γρήγορη σύγκλιση του αλγορίθμου αλλά εγκυμονούν τον κίνδυνο ο αλγόριθμος να ταλαντώνεται γύρω από την τελική λύση , αφού οι αλλαγές των βαρών είναι μεγάλες . Αντίθετα , μικρές τιμές στην παράμετρο a έχουν ως αποτέλεσμα η σύγκλιση του αλγορίθμου να γίνεται μεν πιο αργά αλλά με περισσότερο ομαλό τρόπο .

3.7 Αξιολόγηση της ικανότητας γενίκευσης

Με τον όρο ικανότητα γενίκευσης αναφερόμαστε στην ικανότητα που ευελπιστούμε να έχει το ΤΝΔ μετά την εκπαίδευσή του . Ειδικότερα , αφού το εξειδικεύουμε αρχικά για τα παραδείγματα εκπαίδευσης και στην συνέχεια ευελπιστούμε να εφαρμόσει τις « γνώσεις » που έλαβε σε γενικά παραδείγματα εισόδου (τα οποία είναι παρόμοια με τα παραδείγματα εκπαίδευσης) , τα οποία δεν έχουν χρησιμοποιηθεί κατά την εκπαίδευση . Μπορούμε σε γενικές γραμμές να χωρίσουμε σε δύο κατηγορίες τα προβλήματα τα οποία μπορούν να προκύψουν έπειτα από την εκπαίδευση ενός ΤΝΔ :

1. Τα παραδείγματα εκπαίδευσης να μην είναι αντιπροσωπευτικά του προβλήματος στο οποίο επιθυμούμε να εφαρμόσουμε το ΤΝΔ .
2. Το ΤΝΔ να έχει εκπαιδευθεί πλήρως στα παραδείγματα εκπαίδευσης αλλά η απόδοσή του να μην είναι η αναμενόμενη σε άλλα , παρόμοια παραδείγματα εκπαίδευσης (στην περίπτωση αυτή λέμε ότι το ΤΝΔ έχει υπερ – εκπαιδευθεί , δηλαδή παρουσιάζει καλή απόδοση στα παραδείγματα εκπαίδευσης αλλά δεν είναι σε θέση να γενικεύσει) .

Το πρόβλημα της υπερ - εκπαίδευσης (over-fitting / over-training) παρουσιάζεται διαγραμματικά στην εικόνα 16 . Στο σχήμα αυτό παρουσιάζονται τα παραδείγματα εκπαίδευσης ως μαύρα σημεία . Η συνεχής καμπύλη η οποία διέρχεται από τα σημεία αυτά είναι η συνάρτηση την οποία επιθυμούμε να προσεγγίσουμε ενώ η διακεκομμένη ευθεία γραμμή είναι η συνάρτηση την οποία τελικά το ΤΝΔ προσεγγίζει . Το ΤΝΔ στην περίπτωση αυτή παρουσιάζει καλή απόδοση όσον αφορά το παράδειγμα εκπαίδευσης αλλά δεν προσεγγίζει καθόλου την επιθυμητή συνάρτηση (δεν είναι σε θέση δηλαδή να « γενικεύσει ») .



Εικόνα 16: παράδειγμα υπερ - εκπαίδευσης

Η αξιολόγηση της ικανότητας γενίκευσης ενός ΤΝΔ αποτελεί ένα δύσκολο πρόβλημα . Η δυσκολία οφείλεται στο ότι εξ' αρχής δεν έχουμε στην διάθεσή μας όλες τις δυνατές εισόδους οι οποίες μπορούν να εισαχθούν στο ΤΝΔ παρά μόνο το σύνολο εκπαίδευσης . Φυσικά το σύνολο με τα παραδείγματα εκπαίδευσης ευελπιστούμε να είναι όσο το δυνατόν περισσότερο αντιπροσωπευτικό του πεδίου ορισμού του προβλήματος , καθώς σε διαφορετική περίπτωση το ΤΝΔ θα έχει μικρή εν τέλει ικανότητα γενίκευσης . Αν υποθέσουμε ότι το σύνολο εκπαίδευσης είναι αρκετά αντιπροσωπευτικό του συνολικού πεδίου ορισμού του προβλήματος , τότε μπορούμε να εφαρμόσουμε τεχνικές ελέγχου της ικανότητας γενίκευσης του ΤΝΔ , κάνοντας χρήση μόνο των στοιχείων του συνόλου εκπαίδευσης . Οι κυριότεροι τρόποι χρήσης του συνόλου εκπαίδευσης για την αξιολόγηση της ικανότητας γενίκευσης παρουσιάζονται παρακάτω .

Τεχνική ελέγχου με διακράτηση (holdout)

Η τεχνική αυτή αποτελεί την απλούστερη προσέγγιση . Ειδικότερα , το σύνολο εκπαίδευσης χωρίζεται σε δύο τμήματα , όπου το ένα χρησιμοποιείται για εκπαίδευση και το άλλο για τον έλεγχο της ικανότητας γενίκευσης . Περισσότερο τυπικά , το σύνολο εκπαίδευσης T χωρίζεται σε δύο υποσύνολα $T_{Training}$ και T_{Test} και είναι αρκετά σύνηθες ο πληθάριθμός του $T_{Training}$ να είναι μεγαλύτερος του T_{Test} (δηλαδή $|T_{Training}| > |T_{Test}|$) . Το ΤΝΔ εκπαιδεύεται με την χρήση του $T_{Training}$ και στην συνέχεια κάθε στοιχείο ελέγχου $x_i \in T_{Test}$ εισάγεται ως είσοδος στο ΤΝΔ και εξάγεται η έξοδός του y_i . Κριτήριο ποιότητας του ΤΝΔ αποτελεί η τιμή του σφάλματος $e_i = d_i - y_i$ για κάθε στοιχείο ελέγχου . Είναι αρκετά σύνηθες να υπολογίζεται στην συνέχεια το παρακάτω μέγεθος :

$$MSE = \frac{1}{N}\sum_{i=1}^{N} {e_i}^2$$

όπου $N = |T_{Test}|$.Το παραπάνω μέγεθος ονομάζεται Μέσο Τετραγωνικό Σφάλμα (Mean Squared Error – MSE) και αντιστοιχεί σε έναν θετικό πραγματικό αριθμό . Όσο μικρότερο είναι το MSE τόσο μεγαλύτερη είναι και η ικανότητα γενίκευσης του ΤΝΔ .

Τεχνική ελέγχου με επικύρωση k-τμημάτων (k-fold cross-validation)

Η τεχνική αυτή χωρίζει το σύνολο εκπαίδευσης T σε k μικρότερα υποσύνολα T_1, T_2, …, T_k, με τον ίδιο αριθμό στοιχείων , τα οποία είναι ξένα μεταξύ τους . Στην συνέχεια , για κάθε υποσύνολο T_i εκπαιδεύουμε το ΤΝΔ θεωρώντας ως σύνολο εκπαίδευσης τα υπόλοιπα $k-1$ υποσύνολα και ως σύνολο ελέγχου το T_i . Σε κάθε βήμα εκτέλεσης του αλγορίθμου υπολογίζουμε το MSE_i , για το εκάστοτε σύνολο εκπαίδευσης και εν τέλει υπολογίζουμε το συνολικό σφάλμα ως εξής :

$$Total_{Error} = \sum_{i=1}^{k} MSE_i$$

Η συγκεκριμένη τεχνική ελέγχου είναι πολύ δημοφιλής και η χρήση της για την αξιολόγηση της ικανότητας γενίκευσης αρκετά διαδεδομένη .

Τεχνική LOT (Leave-One-Out)

Η συγκεκριμένη τεχνική αποτελεί ειδική περίπτωση του ελέγχου με επικύρωση k-τμημάτων . Ειδικότερα , κάθε ένα από τα υποσύνολα $T_1, T_2, \ldots, T_k$ αποτελείται από ακριβώς ένα στοιχείο . Η μέθοδος αυτή παρέχει τις πιο αξιόπιστες εκτιμήσεις , ωστόσο έχει το σημαντικό μειονέκτημα του μεγάλου υπολογιστικού φόρτου , αφού η διαδικασία εκπαίδευσης κι ελέγχου , για πρέπει να γίνει για κάθε ένα από τα παραδείγματα εκπαίδευσης .

ΚΕΦΑΛΑΙΟ 4: ΕΞΕΛΙΚΤΙΚΕΣ ΤΕΧΝΙΚΕΣ ΓΙΑ ΤΗΝ ΕΚΠΑΙΔΕΥΣΗ ΤΕΧΝΗΤΩΝ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ

4.1 Εισαγωγή

Στο Κεφάλαιο 3 είδαμε ότι το η εκπαίδευση ενός ΤΝΔ αντιστοιχεί στην εύρεση των σωστών τιμών τις οποίες θα πρέπει να έχουν τα βάρη του . Το πρόβλημα της εκπαίδευσης ενός ΤΝΔ αποτελεί στην ουσία ένα πρόβλημα αναζήτησης , με τον χώρο αναζήτησης να αποτελείται στην ουσία από όλους τους δυνατούς συνδυασμούς τιμών που μπορούν τα βάρη του να πάρουν . Αν εκτός από την εύρεση των βαρών επιθυμούμε και την εύρεση της αρχιτεκτονικής του ΤΝΔ (δηλαδή τον αριθμό των επιπέδων , των νευρώνων σε κάθε επίπεδο αλλά και τις συνδέσεις μεταξύ τους) τότε το ο χώρος αναζήτησης αυξάνει κατά πολύ . Στο Κεφάλαιο 4 θα παρουσιάσουμε τις περισσότερο δημοφιλείς εξελικτικές τεχνικές για την εκπαίδευση ΤΝΔ και θα αναλύσουμε συνοπτικά τον τρόπο λειτουργίας τους .

4.2 Χρήση Γενετικών Αλγορίθμων για την εκπαίδευση Τεχνητών Νευρωνικών Δικτύων

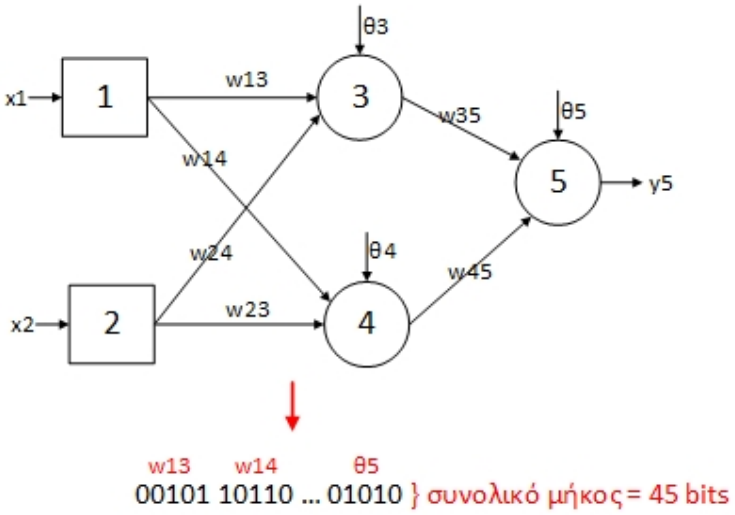
Οι ΓΑ αποτελούν την κατ' εξοχήν εξελικτική τεχνική η οποία μπορεί να εφαρμοστεί για την εκπαίδευση ΤΝΔ . Κύριες προκλήσεις στην εφαρμογή των ΓΑ σε αυτό το πρόβλημα αποτελούν η επιλογή του τρόπου κωδικοποίησης των πιθανών λύσεων ο ορισμός των γενετικών τελεστών της διασταύρωσης και της μετάλλαξης καθώς και ο τρόπος αξιολόγησης της κάθε πιθανής λύσης του πληθυσμού . Οι δημοφιλέστερες παραλλαγές παρουσιάζονται παρακάτω .

4.2.1 Κωδικοποίηση με βάση τις συνδέσεις

Η κωδικοποίηση με βάση τις συνδέσεις αποτελεί τον πλέον απλούστερο τρόπο κωδικοποίησης . Η κωδικοποίηση αυτή μπορεί να εφαρμοστεί μόνο στην περίπτωση όπου η αρχιτεκτονική του ΤΝΔ είναι εξ' αρχής προσδιορισμένη και επιδέχεται μικρές ή καθόλου αλλαγές .

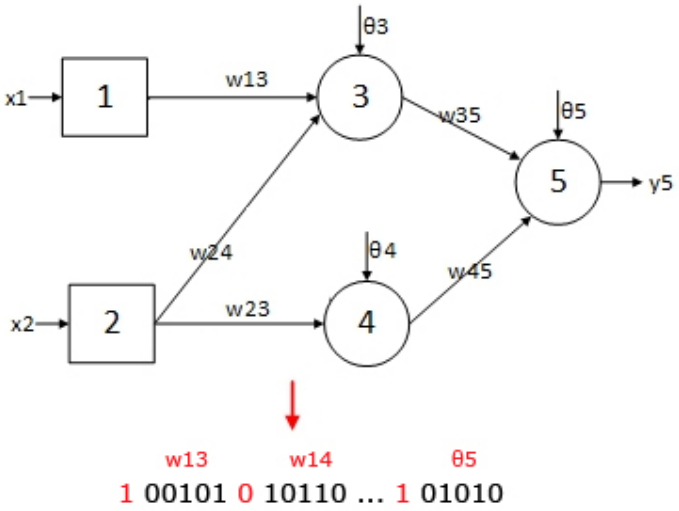
Αν υποθέσουμε ότι το ΤΝΔ συγκεκριμένης αρχιτεκτονικής αποτελείται από n στον αριθμό συνδέσεις , όπου η τιμή της κάθε σύνδεσης κωδικοποιείται από m στον αριθμό bits , τότε μπορούμε να κωδικοποιήσουμε όλα τα βάρη του δικτύου ως μία δυαδική συμβολοσειρά μήκους $n \cdot m$ bits . Με την κωδικοποίηση αυτή , οι κλασικοί δυαδικοί τελεστές της διασταύρωσης και της μετάλλαξης μπορούν να εφαρμοστούν χωρίς καμία παραλλαγή . Ένα παράδειγμα της κωδικοποίησης αυτής παρουσιάζεται στην εικόνα 17 .

Μία διαφορετική προσέγγιση προτάθηκε από τον Montana (Montana 1989) . Ο Montana υπέθεσε ότι η δυαδική κωδικοποίηση σε ΤΝΔ με πολλές συνδέσεις δημιουργεί δυαδικές συμβολοσειρές αρκετά μεγάλου μήκους με αποτέλεσμα την μη αποδοτική χρονικά εφαρμογή των γενετικών τελεστών . Για τον λόγο αυτό πρότεινε την κωδικοποίηση των βαρών του ΤΝΔ με την χρήση πραγματικών αριθμών . Όπως είναι φυσικό , η προσέγγιση αυτή μειώνει κατά πολύ το μήκος της κωδικοποίησης των πιθανών λύσεων . Επίσης δίνει την δυνατότητα επιλογής οποιασδήποτε ακρίβειας στο δεκαδικό μέρος , αφού τυχόν αύξηση των δεκαδικών ψηφίων στην περίπτωση της δυαδικής κωδικοποίησης θα είχε ως αποτέλεσμα την αύξηση του μήκους της δυαδικής συμβολοσειράς . Το κύριο μειονέκτημά της προσέγγισης είναι η μη δυνατότητα άμεσης εφαρμογής των δυαδικών τελεστών της διασταύρωσης και της μετάλλαξης . Για τον λόγο αυτό έπρεπε να οριστούν (Montana 1989) εκ νέου οι τελεστές της διασταύρωσης και της μετάλλαξης , αυξάνοντας χρονικά την απόδοσή τους , καθώς η δυαδική έκδοση των τελεστών αυτών είναι προτιμότερη .



Εικόνα 17: η δυαδική κωδικοποίηση ενός ΤΝΔ με n = 9 συνδέσεις (βάρη) και με m = 5 bits να απαιτούνται για την κωδικοποίηση της κάθε σύνδεσης .

Μία αρκετά δημοφιλής προσέγγιση προτάθηκε από τον Whitley (Whitley 1990) . Η κωδικοποίηση με τον τρόπο αυτόν ονομάζεται GENITOR . Ειδικότερα , η κωδικοποίηση αποτελεί μία παραλλαγή της κωδικοποίησης του περιγράφηκε παραπάνω . Ο αριθμός των νευρώνων στην κωδικοποίηση GENITOR είναι συγκεκριμένος αλλά αυτό που δεν είναι συγκεκριμένο είναι οι μεταξύ τους συνδέσεις . Μπροστά από την τιμή που έχει το κάθε βάρος , εισάγεται επιπρόσθετα το bit 0 ή 1 , το οποίο αντιστοιχεί στην ύπαρξη ή μη του βάρους . Με τον τρόπο αυτό ο ΓΑ είναι σε θέση να αλλάζει ως ένα βαθμό την αρχιτεκτονική του δικτύου , χωρίς όμως να έχει την δυνατότητα προσθήκης ή αφαίρεσης νευρώνων . Ένα παράδειγμα της κωδικοποίησης GENITOR παρουσιάζεται στην εικόνα 18 . Η αρχική έκδοση της μεθόδου η οποία προτάθηκε από τον Whitley χρησιμοποιούσε 8 bits για την τιμή του κάθε βάρους . Κάθε βάρος μπορούσε να πάρει 255 διαφορετικές τιμές (από το -127 μέχρι το +127 με το μηδέν να αναπαρίσταται με δύο διαφορετικές δυαδικές αναπαραστάσεις) .

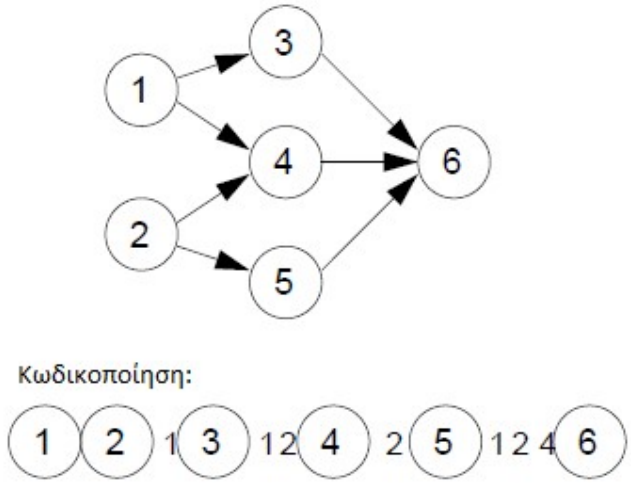


Εικόνα 18: η κωδικοποίηση του παραπάνω ΤΝΔ κάνοντας χρήση της μεθόδου GENITOR . Το βάρος w_{14} απουσιάζει και συνεπώς η τιμή του bit πριν από αυτό είναι ίση με μηδέν .

4.2.3 Κωδικοποίηση με βάση τους κόμβους

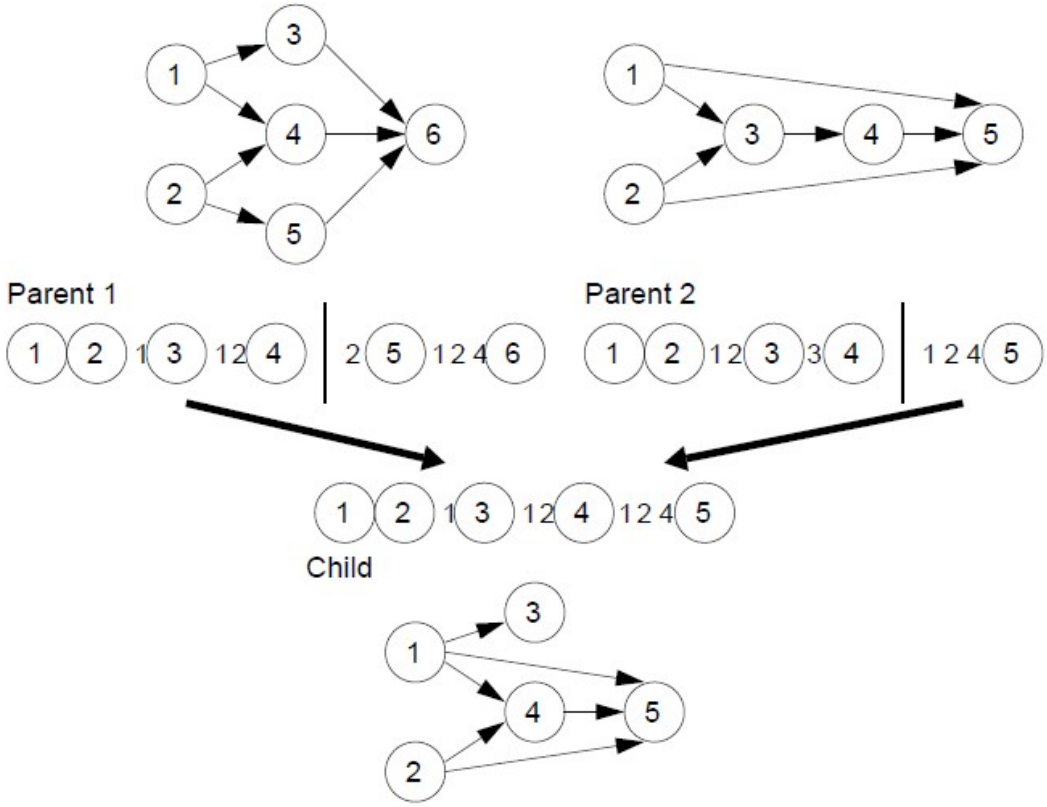
Η κωδικοποίηση αυτής της μορφής είναι κομβοκεντρική . Αυτό πρακτικά σημαίνει ότι περιλαμβάνει όλες τις απαραίτητες πληροφορίες οι οποίες αφορούν ένα νευρώνα , όπως ο αριθμός του κόμβου , το επίπεδο στο οποίο βρίσκεται , οι νευρώνες με τους οποίους συνδέεται , τα βάρη των συνδέσεων κ.α. Κάνοντας χρήση αυτής της κωδικοποίησης , ο ΓΑ είναι πλέον σε θέση εκτός από τα βάρη των συνδέσεων να μεταβάλλει πλέον πλήρως και την αρχιτεκτονική του ΤΝΔ .

Ένα χαρακτηριστικό παράδειγμα κωδικοποίησης με βάση τους κόμβους είναι η κωδικοποίηση του πρότειναν οι Schiffmann , Joost και Werner (Schiffmann 1991 , 1992 , 1993) . Τα γονίδια τα οποία αποτελούν την κωδικοποιημένη συμβολοσειρά είναι ακέραιοι αριθμοί , οι οποίοι αντιστοιχούν στα αναγνωριστικά των νευρώνων του ΤΝΔ . Μπροστά από κάθε αναγνωριστικό υπάρχει μία λίστα με τα αναγνωριστικά των νευρώνων που παρέχουν είσοδο στον τρέχων νευρώνα . Αν επιθυμούμε να αποθηκεύεται και το βάρος της σύνδεσης , δίπλα από κάθε αναγνωριστικό ενός νευρώνα που δίνει είσοδο στον τρέχων νευρώνα , μπορούμε να αποθηκεύουμε και την τιμή του βάρους . Αντίθετα , αν επιθυμούμε να επικεντρωθούμε μόνο στην αρχιτεκτονική του ΤΝΔ , η κωδικοποιημένη συμβολοσειρά αποτελείται μόνο από τα αναγνωριστικά των νευρώνων . Ένα παράδειγμα της κωδικοποίησης αυτής παρουσιάζεται στην εικόνα 19 .



Εικόνα 19: κωδικοποίηση του παραπάνω ΤΝΔ με την χρήση της τεχνικής του Schiffmann . Παρατηρούμε ότι υπάρχουν δύο ειδών γονίδια : οι νευρώνες του ΤΝΔ οι οποίοι αναπαρίστανται με κύκλο και οι είσοδοι του καθενός , οι οποίοι αναπαρίστανται χωρίς κύκλο .

Αυτό το οποίο παρουσιάζει ενδιαφέρον στην παραπάνω προσέγγιση είναι η μέθοδος διασταύρωσης . Η μέθοδος η οποία προτείνεται (Schiffmann 1992 , 1993) αποτελεί μία παραλλαγή της διασταύρωσης μονού σημείου . Ο συγκεκριμένος τελεστής διασταύρωσης διασταυρώνει δύο γονείς και παράγει πάντα έναν απόγονο . Ειδικότερα , αρχικά επιλέγεται ένα σημείο διασταύρωσης το οποίο πρέπει υποχρεωτικά να προηγείται των γονιδίων που αναπαριστούν τις συνδέσεις . Στην συνέχεια ο τελεστής διασταύρωσης ανακτά το τμήμα της συμβολοσειράς που βρίσκεται αριστερά του σημείου διασταύρωσης στον $1^{ο}$ γονέα . Όσον αφορά τον $2^{ο}$ γονέα , ο τελεστής διασταύρωσης ανακτά το τμήμα της συμβολοσειράς που βρίσκεται δεξιά του σημείου διασταύρωσης και αφαιρεί από αυτό όσους νευρώνες βρίσκονται στο αντίστοιχο τμήμα του πρώτου γονέα . Ο απόγονος προκύπτει από την παράθεση των δύο παραπάνω συμβολοσειρών . Στην περίπτωση που στον απόγονο προκύπτουν γονίδια που αναπαριστούν συνδέσεις νευρώνων που πλέον δεν υπάρχουν , τότε τα γονίδια αυτά αφαιρούνται . Όπως είναι προφανές ο απόγονος ο οποίος προκύπτει είναι πιθανό να έχει λιγότερους νευρώνες από τους γονείς του . Ένα παράδειγμα εφαρμογής του τελεστή διασταύρωσης παρουσιάζεται στην εικόνα 20 που ακολουθεί .



Εικόνα 20: ο τελεστής διασταύρωσης ο οποίος προτείνεται στο (Schiffmann 1993, σελ. 678) .

Ειδικά δε για την περίπτωση όπου η αρχιτεκτονική του ΤΝΔ είναι το μόνο ζητούμενο (δηλαδή στην κωδικοποιημένη συμβολοσειρά δεν αποθηκεύονται οι τιμές των βαρών του ΤΝΔ) τότε η αξιολόγηση ενός ατόμου του πληθυσμού γίνεται με τον εξής τρόπο : τα βάρη του ΤΝΔ αρχικοποιούνται με τυχαίες τιμές και το ΤΝΔ εκπαιδεύεται για έναν μικρό αριθμό κύκλων . Το συνολικό λάθος το οποίο προκύπτει μετά την εκπαίδευση, αποτελεί και το κριτήριο ποιότητας της πιθανής λύσης .

Μία διαφορετική προσέγγιση προέρχεται από τον White (White 1993) , ο οποίος έκανε την υπόθεση του ότι η προσθήκη περιορισμών στην αρχιτεκτονική του ΤΝΔ θα οδηγήσει τόσο σε μείωση του χρόνου εκτέλεσης του αλγορίθμου όσο και σε αύξηση της αποδοτικότητας των ΤΝΔ που παράγονται . Η έκδοση αυτή του αλγορίθμου ονομάστηκε από τον White ως GANNET και περιέχει τους ακόλουθους περιορισμούς :

1. Ο αριθμός των εισερχόμενων συνδέσεων (βαρών) ενός νευρώνα είναι ακριβώς 4 .
2. Το ΤΝΔ κατασκευάζεται πάντα με επίπεδα (layers) και για κάθε νευρώνα αποθηκεύεται η πληροφορία για το επίπεδο στο οποίο ανήκει .
3. Για κάθε νευρώνα αποθηκεύεται και οι συνάρτηση ενεργοποίησής του . Υπάρχουν δύο επιτρεπόμενες συναρτήσεις ενεργοποίησης : η σιγμοειδής και η γκαουσιανή . Κατά την αρχικοποίηση του ΤΝΔ , το 80% των νευρώνων του παίρνουν ως συνάρτηση ενεργοποίησης την σιγμοειδή .

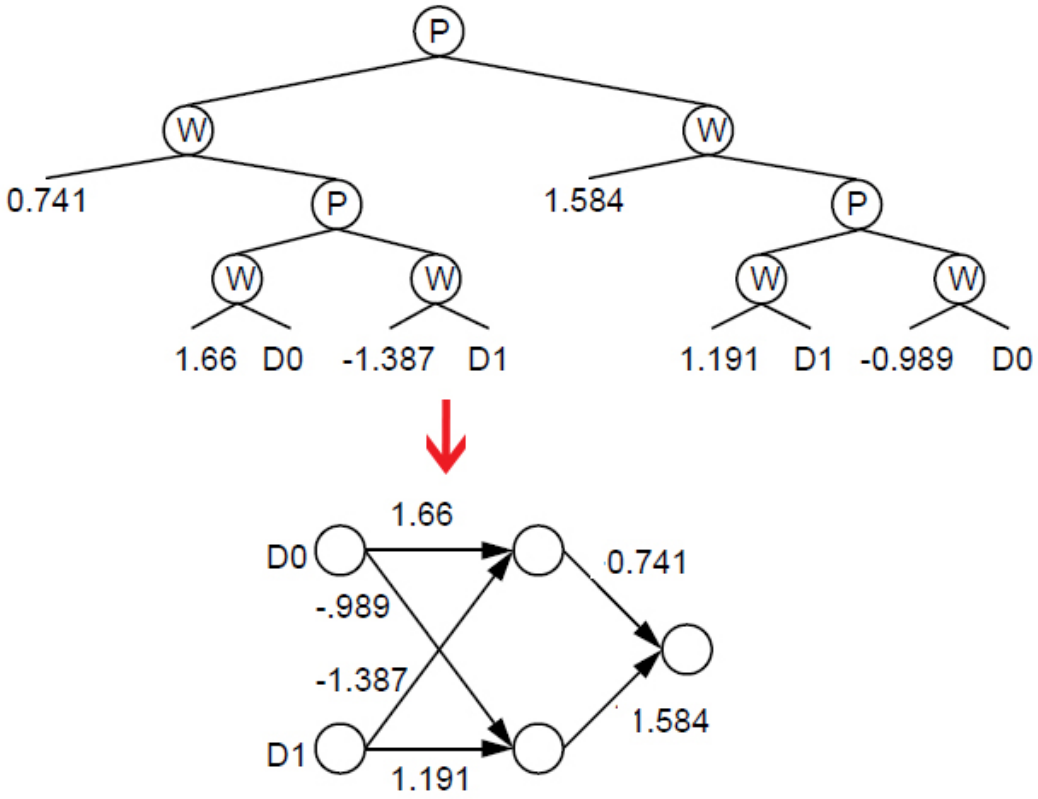
Η δομή της κάθε κωδικοποιημένης συμβολοσειράς είναι παρόμοια με αυτή στα (Schiffmann 1991 , 1992 , 1993) αλλά οι πληροφορίες που αποθηκεύονται για κάθε νευρώνα είναι περισσότερες . Ειδικότερα , για κάθε νευρώνα αποθηκεύουμε τις εξής πληροφορίες :

- Το αναγνωριστικό του νευρώνα .
- Το αναγνωριστικό του επιπέδου (layer) στο οποίο ανήκει .
- Τα αναγνωριστικά των νευρώνων από τα οποία δέχεται είσοδο (4 ακριβώς) .

- Τις τιμές των 5 εισερχόμενων βαρών του νευρώνα (4 από τις μεταβλητές συνάψεις και 1 λόγω του κατωφλίου) .
- Την συνάρτηση ενεργοποίησης του νευρώνα (σιγμοειδής ή γκαουσιανή) .

Ο τελεστής διασταύρωσης είναι παρόμοιος με αυτόν που περιγράφεται στα (Schiffmann 1992 , 1993) . Όσον αφορά τον τελεστή μετάλλαξης , αυτός επιλέγει τυχαία ένα ή περισσότερα βάρη ενός νευρώνα και μεταβάλλει τις τιμές τους .

Μία εντελώς διαφορετική προσέγγιση προτάθηκε από τον Koza (Koza 1991) , ο οποίος κωδικοποίησε κάθε πιθανή λύση ως ένα δέντρο , το οποίο έχει δύο ειδών εσωτερικούς κόμβους : υπολογιστικούς νευρώνες και βάρη . Τα παιδιά ενός υπολογιστικού νευρώνα είναι τα εισερχόμενα βάρη του ενώ τα παιδιά ενός βάρους είναι αφενός η τιμή του βάρους και αφετέρου ο νευρώνας από τον οποίον το βάρος εξέρχεται . Τα φύλλα του δέντρου είναι οι τιμές των βαρών καθώς και οι νευρώνες εισόδου , οι οποίοι δεν έχουν εισερχόμενα βάρη . Ένα παράδειγμα της κωδικοποίησης αυτής παρουσιάζεται στην εικόνα 21 .



Εικόνα 21: η αναπαράσταση ενός ΤΝΔ με την μορφή δέντρου . Οι εσωτερικοί κόμβοι με την ονομασία P αντιστοιχούν σε υπολογιστικούς νευρώνες ενώ οι εσωτερικοί κόμβοι με την ονομασία W αντιστοιχούν σε βάρη . Οι τερματικοί κόμβοι του δέντρου με την ονομασία D αντιστοιχούν σε νευρώνες εισόδου , οι οποίοι δεν διαθέτουν εισερχόμενα βάρη .

Ο τελεστής διασταύρωσης ο οποίος χρησιμοποιείται ανταλλάσσει κόμβους μεταξύ δύο δέντρων . Φυσικά κατά την διαδικασία της διασταύρωσης θα πρέπει να τηρείται ένα σύνολο περιορισμών , έτσι ώστε οι απόγονοι να αποτελούν έγκυρα δέντρα (να έχουν δηλαδή την δομή που περιγράφηκε παραπάνω) . Όσον αφορά τον τελεστή μετάλλαξης , αυτός έχει δύο λειτουργίες :

1. Ανταλλάσσει υπόδεντρα μέσα στο ίδιο δέντρο .
2. Αλλάζει με τυχαίο τρόπο την τιμή ενός ή περισσότερων βαρών .

4.2.3 Αξιολόγηση των πιθανών λύσεων

Η αξιολόγηση των πιθανών λύσεων αποτελεί ένα από τα περισσότερο κομβικά σημεία του ΓΑ στο συγκεκριμένο πεδίο προβλήματος . Η συνάρτηση αξιολόγησης θα πρέπει προφανώς να λαμβάνει υπ' όψη το συνολικό λάθος που υπάρχει στις εξόδους του ΤΝΔ για κάθε ένα από τα παραδείγματα εκπαίδευσης , τα οποία χρησιμοποιούνται ως κριτήριο ελέγχου της καταλληλότητας . Μία αρκετά συνήθης τακτική είναι ο έλεγχος των παραδειγμάτων εκπαίδευσης όχι απευθείας σε μία πιθανή λύση , αλλά μετά την εκπαίδευσή της για έναν συγκεκριμένο αριθμό κύκλων εκπαίδευσης με την χρήση του αλγορίθμου Οπισθοδιάδοσης του Λάθους . Η λογική πίσω από την προσέγγιση αυτή κρύβεται στο ότι η ποιότητα του ΤΝΔ το οποίο αναπαριστά η πιθανή λύση δεν είναι δυνατό να φανεί από ένα και μόνο στιγμιότυπο στις τιμές των βαρών του . Συνεπώς αν το ΤΝΔ εκπαιδευθεί για έναν , μικρό σε γενικές γραμμές , αριθμό κύκλων , ο αλγόριθμος θα είναι σε θέση να έχει μία καλύτερη εικόνα για την ποιότητά του .

4.3 Χρήση του αλγορίθμου PSO για την εκπαίδευση Τεχνητών Νευρωνικών Δικτύων

Παρά το γεγονός του ότι οι Γενετικοί Αλγόριθμοι αποτελούν μία κοινώς αποδεκτή λύση για την επίλυση του προβλήματος της εκπαίδευσης ΤΝΔ εμπρόσθιας τροφοδότησης , λόγω της ικανότητάς τους να αναζητούν αποδοτικά μεγάλους χώρους αναζήτησης , παρουσιάζουν παράλληλα κάποια μειονεκτήματα . Ειδικότερα , λόγω της εφαρμογής του τελεστή της μετάλλαξης είναι πιθανό να υπάρξει γρήγορη σύγκλιση του ΓΑ σε κάποιο τοπικό βέλτιστο ενώ θα χρειαζόταν περισσότερο χρόνο για να προσεγγίσει το ολικό βέλτιστο του χώρου αναζήτησης . Σε αντίθεση με τον ΓΑ , ο αλγόριθμος PSO συνήθως εκτελείται σε περισσότερο χρόνο και αυτό αυξάνει τις πιθανότητας κάλυψης μεγαλύτερου τμήματος στον χώρο αναζήτησης . Για την ακρίβεια έχει αποδειχθεί πειραματικά ότι σε ορισμένα προβλήματα ο αλγόριθμος PSO υπερτερεί του ΓΑ ως προς την ποιότητα της επιστρεφόμενης λύσης (Eberhart, Shi 1998) . Αυτό οφείλεται κυρίως στο γεγονός του ότι ο αλγόριθμος PSO διαθέτει σημαντικά χαρακτηριστικά όπως επικοινωνία μεταξύ των μορίων του πληθυσμού , με αποτέλεσμα αυτά να αποφεύγουν τυχόν παθογένειες που μπορεί να προκύψουν από τον τελεστή της μετάλλαξης στην περίπτωση του ΓΑ .

Έχει προταθεί ένα σύνολο διαφορετικών παραλλαγών του αλγορίθμου PSO για την εκπαίδευση ΤΝΔ εμπρόσθιας τροφοδότησης. Όπως και στους ΓΑ, έτσι και στην περίπτωση του αλγορίθμου PSO , ο όρος εκπαίδευση μπορεί να θεωρηθεί με τους εξής δύο εναλλακτικούς τρόπους:

1. Τροποποίηση μόνο των βαρών του ΤΝΔ , ενώ οι άλλες παράμετροι (αριθμός κρυφών επιπέδων , αριθμός νευρώνων σε κάθε επίπεδο , συνδέσεις μεταξύ των νευρώνων και συναρτήσεις ενεργοποίησης) είναι ήδη καθορισμένες .
2. Ταυτόχρονη εξέλιξη της δομής του ΤΝΔ με τις τιμές των βαρών .

Επειδή ο αλγόριθμος PSO κωδικοποιεί τις πιθανές λύσεις ως διανύσματα πραγματικών τιμών , η κωδικοποίηση των υπόλοιπων παραμέτρων ενός ΤΝΔ , εκτός των βαρών , αποτελεί ένα δύσκολο πρόβλημα . Αν επιθυμούμε να κωδικοποιήσουμε και τις υπόλοιπες παραμέτρους του ΤΝΔ αυτό θα δημιουργήσει την ανάγκη για την εισαγωγή ελέγχων εγκυρότητας των πιθανών λύσεων που προκύπτουν σε κάθε κύκλο εκπαίδευσης καθώς και την εισαγωγή μεθόδων διόρθωσης σε τυχόν μη εγκυρότητα κάποιων λύσεων . Οι ενέργειες αυτές θα έχουν ως αποτέλεσμα την αύξηση του χρόνου εκτέλεσης του αλγορίθμου και εν τέλει την μη αποδοτικότητά του . Για τον λόγο αυτό ο αλγόριθμος PSO είναι περισσότερο κατάλληλος για την εκπαίδευση ΤΝΔ προκαθορισμένης δομής .

Η εκπαίδευση ΤΝΔ προκαθορισμένης δομής έχει μελετηθεί από τον Mendes (Mendes et all, 2002) . Τα στοιχεία του κάθε διανύσματος του αλγορίθμου PSO αποτελούν τα βάρη του ΤΝΔ εμπρόσθιας τροφοδότησης . Η έρευνά του έδειξε ότι ο αλγόριθμος PSO παρουσιάζει καλή απόδοση σε χώρους αναζήτησης όπου αποδεδειγμένα υπάρχουν πολλά τοπικά βέλτιστα , κάτι το οποίο έρχεται να επιβεβαιώσει τις υποθέσεις τις οποίες κάναμε παραπάνω .

Μία παραλλαγή του PSO η οποία θέτει περιορισμό στην μέγιστη ταχύτητα την οποία μπορεί να αποκτήσει ένα μόριο προτάθηκε και μελετήθηκε από τον Xiaorong (Xiaorong et all, 2007) . Χρησιμοποιώντας ένα άνω όριο ταχύτητας μειώνουμε την πιθανότητας πρόωρης σύγκλισης του

αλγορίθμου και ταυτόχρονα αποτρέπουμε το σωματίδιο από το να ξεπεράσει του χώρου αναζήτησης (έτσι ώστε να μην δημιουργηθούν μη έγκυρες πιθανές λύσεις) .

Μία τολμηρή παραλλαγή του αλγορίθμου PSO προτάθηκε από τους Van den Bergh και Engelbrecht (Van den Bergh , Engelbrecht , 2004) . Στην παραλλαγή αυτή , το κάθε διάνυσμα βαρών χωρίζεται σε μικρό υπο-διανύσματα, όπου το κάθε ένα από αυτά αλλάζει τις τιμές του από διαφορετικό σμήνος . Η συγκεκριμένη έκδοση του αλγορίθμου μειώνει τον συνολικό χρόνο εκτέλεσης αφού με κατάλληλη υλοποίηση μπορεί να εκτελεστεί σε ένα παράλληλο σύστημα .

Μία έκδοση του αλγορίθμου PSO η οποία θυμίζει την ακριβώς παραπάνω , είναι αυτή των Al-Kazemi και Mohan (Al-kazemi, Mohan 2002) . Η έκδοση αυτή ονομάστηκε MPPSO (Multi - Phase PSO) και η εκτέλεση του αλγορίθμου χωρίζεται σε χρονικές φάσεις . Ο πληθυσμός αποτελείται και εδώ από υπο - ομάδες και η κάθε ομάδα αλλάζει την κατεύθυνση των υπόλοιπων ομάδων σε κάθε διαφορετική φάση του αλγορίθμου . Η προσέγγιση αυτή έχει το πλεονέκτημα της εκτενέστερης κάλυψης του χώρου αναζήτησης . Θα πρέπει επίσης να τονιστεί ότι οι συναρτήσεις αλλαγής της ταχύτητας και της θέσης των μορίων είναι διαφορετικές από την τυπική έκδοση του αλγορίθμου , αφού θα πρέπει να υλοποιούν την συγκεκριμένη συμπεριφορά.

Μία τέλος αρκετά αποδοτική προσέγγιση είναι αυτή η οποία συνδυάζει την εκτέλεση του αλγορίθμου PSO με την εκτέλεση των ΓΑ . Ο συγκεκριμένος υβριδικός αλγόριθμος ονομάζεται HGAPSO (Hybrid Genetic Algorithm and Particle Swarm Optimization) και όσον αφορά την εκπαίδευση ΤΝΔ προτάθηκε από τον Juang . Στην μέθοδο HGAPSO τα νέα άτομα του πληθυσμού δεν δημιουργούνται μόνο με την χρήση των τελεστών της διασταύρωσης και της μετάλλαξης του ΓΑ αλλά γίνεται επίσης χρήση των συναρτήσεων του αλγορίθμου PSO . Στην προσέγγιση του Juang , ο μισός πληθυσμός εξελίσσεται με την χρήση των τελεστών του ΓΑ ενώ ο υπόλοιπος μισός με την χρήση των συναρτήσεων του PSO .

ΚΕΦΑΛΑΙΟ 5: ΜΙΑ ΥΒΡΙΔΙΚΗ ΕΞΕΛΙΚΤΙΚΗ ΤΕΧΝΙΚΗ ΓΙΑ ΤΗΝ ΕΚΠΑΙΔΕΥΣΗ ΤΕΧΝΗΤΩΝ ΝΕΥΡΩΝΙΚΩΝ ΔΙΚΤΥΩΝ

5.1 Εισαγωγή

Στο Κεφάλαιο 5 θα παρουσιάσουμε μία υβριδική εξελικτική τεχνική για την εκπαίδευση ΤΝΔ . Ειδικότερα, η προσέγγισή μας θα συνδυάζει τα χαρακτηριστικά τόσο του Γενετικού Αλγορίθμου όσο και της Βελτιστοποίηση με Νοημοσύνη Σμήνους . Η πρότασή μας ονομάζεται Υβριδικός Γενετικός Αλγόριθμος με την χρήση Βελτιστοποίησης Νοημοσύνης Σμήνους (Hybrid Genetic Algorithm - Particle Swarm Optimization , HGAPSO) και θα χρησιμοποιηθεί για την εκπαίδευση ΤΝΔ τα οποία είναι συγκεκριμένης αρχιτεκτονικής . Ο αλγόριθμος HGAPSO θα εκτελεί για έναν αριθμό γενεών τον ΓΑ και στην συνέχεια στον πληθυσμό που θα προκύψει θα εκτελεί τον αλγόριθμο PSO .

5.2 Ορισμός του Υβριδικού Γενετικού Αλγορίθμου με την χρήση Βελτιστοποίησης Νοημοσύνης Σμήνους

Ο Υβριδικός Γενετικός Αλγόριθμος με την χρήση Βελτιστοποίησης Νοημοσύνης Σμήνους (Hybrid Genetic Algorithm-Particle Swarm Optimization , HGAPSO) συνδυάζει τόσο τα πλεονεκτήματα των ΓΑ όσο και του αλγορίθμου PSO . Η κύρια λογική πίσω από την λειτουργία του είναι η εξής :

1. Ο ΓΑ χρησιμοποιείται για την εξέλιξη του πληθυσμού για έναν αριθμό γενεών x .
2. Στην συνέχεια ο αλγόριθμος PSO συνεχίζει την εξέλιξη του πληθυσμού για έναν αριθμό γενεών y .

Έτσι λοιπόν αν $round$ είναι ο συνολικός αριθμός γενεών όπου ο αλγόριθμος HGAPSO θα εκτελεστεί , ισχύει ότι $x + y = rounds$. Επίσης μπορούμε να εκφράσουμε τα x, y ως ποσοστά του αριθμού των γενεών $rounds$, αφού $x = a \cdot rounds$ και $y = b \cdot rounds$, με $0 < a, b < 1$ και $a + b = 1$. Τα ακριβή βήματα του αλγορίθμου HGAPSO παρουσιάζονται παρακάτω .

5.3 Περιγραφή του Γενετικού Αλγορίθμου

Στο τμήμα αυτό θα περιγράψουμε την δομή του ΓΑ , ο οποίος αποτελεί το πρώτο τμήμα του HGAPSO . Ειδικότερα , θα ασχοληθούμε με τα εξής θέματα :

1. Επιλογή κωδικοποίησης
2. Διαδικασία αρχικοποίησης του πληθυσμού
3. Αξιολόγηση των πιθανών λύσεων
4. Τελεστής επιλογής
5. Τελεστής διασταύρωσης
6. Τελεστής μετάλλαξης

5.3.1 Επιλογή κωδικοποίησης

Ο αλγόριθμος HGAPSO ασχολείται με την εύρεση των βαρών ενός ΤΝΔ συγκεκριμένης αρχιτεκτονικής . Το ΤΝΔ μπορεί να έχει οποιαδήποτε αρχιτεκτονική ενώ τα βάρη του μπορούν να λάβουν οποιαδήποτε πραγματική τιμή στο διάστημα $[-10,10]$ με ακρίβεια το πολύ 5 δεκαδικά ψηφία . Η κωδικοποίηση η οποία χρησιμοποιεί ο ΓΑ χρησιμοποιεί είναι η δυαδική κωδικοποίηση . Οι παραπάνω παράμετροι έχουν ως αποτέλεσμα το κάθε βάρος να κωδικοποιείται με μία δυαδική συμβολοσειρά μήκους 21 bits . Συνεπώς αν ο αριθμός των βαρών του ΤΝΔ είναι wn , τότε το συνολικό μήκος της δυαδικής συμβολοσειράς ισούται με $21 \cdot wn$ bits . Η κωδικοποίηση και η αποκωδικοποίηση των ατόμων του πληθυσμού γίνεται με τους τύπους που περιγράφηκαν στην ενότητα 2.2.1 .

5.3.2 Διαδικασία αρχικοποίησης του πληθυσμού

Η διαδικασία αρχικοποίησης του πληθυσμού POP μεγέθους POP_SIZE λαμβάνει χώρα με βάση τον παρακάτω αλγόριθμο :

```
POP := { }
Για i από 1 έως POP_SIZE επανάλαβε
   Υπολόγισε μία τυχαία πραγματική τιμή r_i ∈ [−10,10] με ακρίβεια 5 δεκαδικών ψηφίων
   Κωδικοποίησε την r_i βάσει των τύπων της ενότητας 2.2.1
   Ονόμασε την κωδικοποιημένη δυαδική συμβολοσειρά a_i
   Πρόσθεσε την a_i στον πληθυσμό POP
Τέλος_Επανάληψης
```

5.2.3 Αξιολόγηση των πιθανών λύσεων

Η αξιολόγηση των πιθανών λύσεων γίνεται κάνοντας χρήση των δεδομένων εκπαίδευσης . Ειδικότερα , το κάθε άτομο του πληθυσμού αντιστοιχεί σε ένα ΤΝΔ . Το άτομο αποκωδικοποιείται στις τιμές των βαρών και το ΤΝΔ κατασκευάζεται . Στην συνέχεια πραγματοποιείται εμπρόσθιο πέρασμα για όλα τα δεδομένα εισόδου και υπολογίζεται το Μέσο Τετραγωνικό Σφάλμα (Mean Squared Error – MSE) . Η συνάρτηση αξιολόγησης για ένα άτομο του πληθυσμού a_i είναι η παρακάτω :

$$eval(a_i) = \frac{1}{MSE} \cdot 100$$

Η συνάρτηση αξιολόγησης λαμβάνει αυτή την μορφή επειδή η εκπαίδευση ενός ΤΝΔ πρόκειται για πρόβλημα ελαχιστοποίησης του MSE και όχι μεγιστοποίησής του . Η τιμή 1/MSE πολλαπλασιάζεται με το 100 , αφού είναι αρκετά σύνηθες το 1/MSE να οδηγεί σε αρκετά μικρές τιμές, ειδικά αν το MSE είναι μεγάλο (κάτι το οποίο ίσως δημιουργούσε πρόβλημα στους υπολογισμούς των επόμενων βημάτων) .

5.2.4 Τελεστής επιλογής

Ο ΓΑ κάνει χρήση του τελεστή επιλογής της εξαναγκασμένης ρουλέτας , με μία μικρή παραλλαγή . Ειδικότερα , έχει προβλεφθεί η περίπτωση όπου το MSE για ένα άτομο του πληθυσμού είναι ίσο με το μηδέν (αυτό πρακτικά σημαίνει ότι το ΤΝΔ απαντά σωστά σε όλα τα δεδομένα εκπαίδευσης , χωρίς κανένα σφάλμα) . Στην περίπτωση αυτή , επειδή ο υπολογισμός της καταλληλότητας είναι αδύνατος , ο τελεστής επιλογής επιλύει αυτό το πρόβλημα ως εξής : εντοπίζει το άτομο του πληθυσμού με το μικρότερο MSE και αποδίδει στα άτομα με MSE = 0 την τιμή MSE/10 . Αυτό έχει ως αποτέλεσμα η συνάρτηση καταλληλότητας $eval(a_i)$ να είναι πλέον σε θέση να υπολογιστεί και να αποδίδει μεγαλύτερες τιμές στα άτομα με MSE = 0 .

5.2.5 Τελεστής διασταύρωσης

Ο τελεστής διασταύρωσης ο οποίος χρησιμοποιείται είναι ο κλασικός τελεστής διασταύρωσης μονού σημείου , με μία επίσης παραλλαγή . Ειδικότερα , μετά την δημιουργία των απογόνων ενός ζεύγους γονέων , αυτοί ελέγχονται για την εγκυρότητά τους (η αποκωδικοποιημένη τους μορφή θα πρέπει να είναι ένας πραγματικός αριθμός ακρίβειας 5 δεκαδικών ψηφίων στο διάστημα $[-10,10]$) . Σε περίπτωση όπου αυτό δεν ισχύει , τότε η απόγονοι λαμβάνουν τυχαία μία τιμή στο διάστημα αυτό .

5.2.6 Τελεστής μετάλλαξης

Ο τελεστής μετάλλαξης ο οποίος χρησιμοποιείται είναι και αυτός ο κλασικός τελεστής μετάλλαξης που παρουσιάστηκε στην ενότητα 2.2.1 . Και εδώ όμως, όπως και στον τελεστή διασταύρωσης , μετά την εκτέλεση του τελεστή μετάλλαξης σε ένα άτομο , αυτό ελέγχεται για την εγκυρότητά του . Έτσι λοιπόν, αν δεν πρόκειται για έναν πραγματικό αριθμό ακρίβειας 5 δεκαδικών ψηφίων στο διάστημα $[-10,10]$, τότε αντικαθίσταται με μία τυχαία τιμή στο παραπάνω διάστημα .

5.4 Περιγραφή του αλγορίθμου PSO

Ο αλγόριθμος PSO εκτελείται αμέσως μετά την εκτέλεση του ΓΑ . Ειδικότερα , ο αλγόριθμος PSO δεν εκτελεί την διαδικασία της αρχικοποίησης αλλά χρησιμοποιεί τον πληθυσμό που προκύπτει από τον ΓΑ και συνεχίζει την εξέλιξη σε αυτόν . Το κάθε μόριο του σμήνους είναι ένα πραγματικό διάνυσμα με διάσταση ίση με τον αριθμό των βαρών του ΤΝΔ . Επειδή η κωδικοποίηση των πιθανών λύσεων δεν είναι η δυαδική , στον αλγόριθμο PSO δεν πραγματοποιούνται διορθώσεις στις τιμές των βαρών , στην περίπτωση όπου κάποιο από αυτό βγει εκτός των ορίων του διαστήματος $[-10,10]$. Το ΤΝΔ που αντιστοιχεί στο μόριο του σμήνους με την καλύτερη αξιολόγηση στον τελευταίο κύκλο εκπαίδευσης είναι και αυτό το οποίο επιστρέφεται ως τελική λύση από τον αλγόριθμο HGAPSO .

ΚΕΦΑΛΑΙΟ 6: ΜΕΘΟΔΟΛΟΓΙΑ ΣΥΓΚΡΙΣΗΣ ΤΩΝ ΑΛΓΟΡΙΘΜΩΝ ΕΚΠΑΙΔΕΥΣΗΣ ΤΝΔ

6.1 Εισαγωγή

Στο Κεφάλαιο αυτό θα παρουσιάσουμε την μεθοδολογία την οποία ακολουθήσαμε για την σύγκριση της απόδοσης των εξελικτικών αλγορίθμων . Ειδικότερα , θα εκτελέσουμε τα παρακάτω βήματα :

1. Υλοποίηση λογισμικού το οποίο θα είναι σε θέση να δέχεται ως είσοδο δεδομένα εκπαίδευσης και να εκπαιδεύει ένα ΤΝΔ συγκεκριμένης αρχιτεκτονικής με την χρήση των παρακάτω αλγορίθμων :
    a. Αλγόριθμος Οπισθοδιάδοσης του Λάθους
    b. Γενετικός Αλγόριθμος
    c. Αλγόριθμος PSO
    d. Αλγόριθμος HGAPSO
2. Εύρεση δεδομένων εκπαίδευσης τα οποία θα χρησιμοποιηθούν από το λογισμικό .
3. Εύρεση των βέλτιστων τιμών για τις παραμέτρους a, b του αλγορίθμου HGAPSO .
4. Εκτέλεση των 4 αλγορίθμων και καταγραφή των πειραματικών αποτελεσμάτων .

Τα βήματα τα οποία θα ακολουθήσουμε περιγράφονται αναλυτικά παρακάτω .

6.2 Υλοποίηση λογισμικού

Το λογισμικό το οποίο υλοποιήθηκε για την εκτέλεση των πειραμάτων περιγράφεται αναλυτικά στο Κεφάλαιο 7 που ακολουθεί . Το λογισμικό υλοποιήθηκε στην γλώσσα προγραμματισμού Java και πληροί τις παρακάτω προδιαγραφές :

- Δέχεται ως είσοδο αρχεία XML , τα οποία περιγράφουν τα δεδομένα εκπαίδευσης .
- Δίνει στον χρήστη την δυνατότητα παραμετροποίησης του ΤΝΔ (αριθμός επιπέδων και αριθμός νευρώνων ανά επίπεδο) .
- Υλοποιεί τους 4 αλγόριθμους εκπαίδευσης και δίνει την δυνατότητα στον χρήστη να θέσει τιμές στις παραμέτρους αυτών .
- Υπολογίζει το Μέσο Τετραγωνικό Σφάλμα (MSE) για κάθε ΤΝΔ το οποίο προκύπτει κατά την διαδικασία της εκπαίδευσης .

6.3 Δεδομένα εκπαίδευσης

Για την εκπαίδευση των ΤΝΔ χρησιμοποιήθηκαν 5 δεδομένα εκπαίδευσης , τα οποία ανακτήθηκαν από το UCI Machine Learning Repository (https://archive.ics.uci.edu/ml/datasets.html) . Τα δεδομένα εκπαίδευσης μετασχηματίστηκαν κατάλληλα, σε ισοδύναμη μορφή, έτσι ώστε να είναι δυνατός ο χειρισμός τους από το λογισμικό . Ειδικότερα , οι μετασχηματισμοί οι οποίοι πραγματοποιήθηκαν είναι οι εξής :

1. Τα δεδομένα εκπαίδευσης τοποθετήθηκαν μέσα σε XML αρχεία , η δομή των οποίων παρουσιάζεται στο επόμενο κεφάλαιο .
2. Τα δεδομένα κάθε συνόλου δεδομένων μετατράπηκαν σε αριθμητικά . Μάλιστα , επειδή οι νευρώνες του ΤΝΔ θα κάνουν χρήση της σιγμοειδούς συνάρτησης ως συνάρτηση ενεργοποίησης , οι επιθυμητές έξοδοι κανονικοποιήθηκαν στο διάστημα $[0,1]$.

Τα χαρακτηριστικά του καθενός από τα σύνολα δεδομένων εκπαίδευσης παρουσιάζονται παρακάτω .

6.3.1 Σύνολο δεδομένων εκπαίδευσης Abalone

Το σύνολο δεδομένων εκπαίδευσης Abalone (https://archive.ics.uci.edu/ml/datasets/Abalone) αφορά την συσχέτιση μεταξύ χαρακτηριστικών και της ηλικίας μιας κατηγορίας οστρακοειδών . Τα χαρακτηριστικά του παρουσιάζονται στον Πίνακα 1 που ακολουθεί .

Σύνολο Δεδομένων: Abalone								
Αριθμός δεδομένων εκπαίδευσης: 4177								
Αριθμός Εισόδων: 8 / Αριθμός Εξόδων: 1								
Είσοδοι								**Έξοδος**
Φύλο	Μήκος	Διάμετρος	Ύψος	Συνολικό Βάρος	Καθαρό Βάρος	Βάρος Κρέατος	Βάρος Κελύφους	Ηλικία
M, F	Πραγματικός	Πραγματικός	Πραγματικός	Πραγματικός	Πραγματικός	Πραγματικός	Πραγματικός	Ακέραιος

Πίνακας 1: χαρακτηριστικά του συνόλου δεδομένων Abalone.

6.3.2 Σύνολο δεδομένων εκπαίδευσης Balance-Scale

Το σύνολο δεδομένων εκπαίδευσης Balance-Scale (https://archive.ics.uci.edu/ml/datasets/Balance+Scale) αφορά την συσχέτιση μεταξύ χαρακτηριστικών και της ψυχικής ισορροπίας ατόμων . Τα χαρακτηριστικά του παρουσιάζονται στον Πίνακα 2 που ακολουθεί .

Σύνολο Δεδομένων: Balance-Scale				
Αριθμός δεδομένων εκπαίδευσης: 625				
Αριθμός Εισόδων: 4 / Αριθμός Εξόδων: 1				
Είσοδοι				**Έξοδος**
Αριστερό Βάρος	Αριστερή Απόσταση	Δεξιό Βάρος	Δεξιά Απόσταση	Ψυχική Ισορροπία
1, 2, 3, 4, 5	1, 2, 3, 4, 5	1, 2, 3, 4, 5	1, 2, 3, 4, 5	L, B, R

Πίνακας 2: χαρακτηριστικά του συνόλου δεδομένων Balance-Scale .

6.3.3 Σύνολο δεδομένων εκπαίδευσης Car Evaluation

Το σύνολο δεδομένων εκπαίδευσης Car Evaluation (https://archive.ics.uci.edu/ml/datasets/Car+Evaluation) αφορά την συσχέτιση μεταξύ χαρακτηριστικών και της ποιότητας αυτοκινήτων. Τα χαρακτηριστικά του παρουσιάζονται στον Πίνακα 3 που ακολουθεί .

Σύνολο Δεδομένων: Car Evaluation						
Αριθμός δεδομένων εκπαίδευσης: 1728						
Αριθμός Εισόδων: 6 / Αριθμός Εξόδων: 1						
Είσοδοι						**Έξοδος**
Τιμή	Συντήρηση	Αριθμός Θυρών	Άτομα	Χωρητικότητα	Επίπεδο Προστασίας	Ποιότητα
vhigh, high, med, low	vhigh, high, med, low	2, 3, 4, 5more	2, 4, more	small, med, big	low, med, high	Ακέραιος

Πίνακας 3: χαρακτηριστικά του συνόλου δεδομένων Car Evaluation .

6.3.4 Σύνολο δεδομένων εκπαίδευσης Cloud

Το σύνολο δεδομένων εκπαίδευσης Cloud (https://archive.ics.uci.edu/ml/datasets/Cloud) αφορά την συσχέτιση μεταξύ μετεωρολογικών μετρήσεων που αφορούν νέφη και της ορατότητας . Τα χαρακτηριστικά του παρουσιάζονται στον Πίνακα 4 που ακολουθεί .

Σύνολο Δεδομένων: Cloud							
Αριθμός δεδομένων εκπαίδευσης: 1024							
Αριθμός Εισόδων: 7 / Αριθμός Εξόδων: 3 (χρησιμοποιούμε την μία από αυτές)							
Είσοδοι							**Έξοδος**
mean	max	min	mean distribution	contrast	entropy	second angular momentum	Ορατότητα
Πραγματικός	Πραγματικός	Πραγματικός	Πραγματικός	Πραγματικός	Πραγματικός	Πραγματικός	Πραγματικός

Πίνακας 4: χαρακτηριστικά του συνόλου δεδομένων Cloud .

6.3.5 Σύνολο δεδομένων εκπαίδευσης Iris

Το σύνολο δεδομένων εκπαίδευσης Iris (https://archive.ics.uci.edu/ml/datasets/Iris) αφορά την συσχέτιση μεταξύ χαρακτηριστικών του φυτού Ίρις και των υποκατηγοριών στις οποίες χωρίζεται . Τα χαρακτηριστικά του παρουσιάζονται στον Πίνακα 5 που ακολουθεί .

Σύνολο Δεδομένων: Iris				
Αριθμός δεδομένων εκπαίδευσης: 352				
Αριθμός Εισόδων: 4 / Αριθμός Εξόδων: 1				
Είσοδοι				**Έξοδος**
Μήκος άνθους	Πλάτος άνθους	Μήκος πετάλου	Πλάτους πετάλου	Κατηγορία
Πραγματικός	Πραγματικός	Πραγματικός	Πραγματικός	Iris Setosa, Iris Versicolour, Iris Virginica

Πίνακας 5: χαρακτηριστικά του συνόλου δεδομένων Iris .

6.4 Εύρεση των βέλτιστων τιμών για τις παραμέτρους a, b του αλγορίθμου HGAPSO

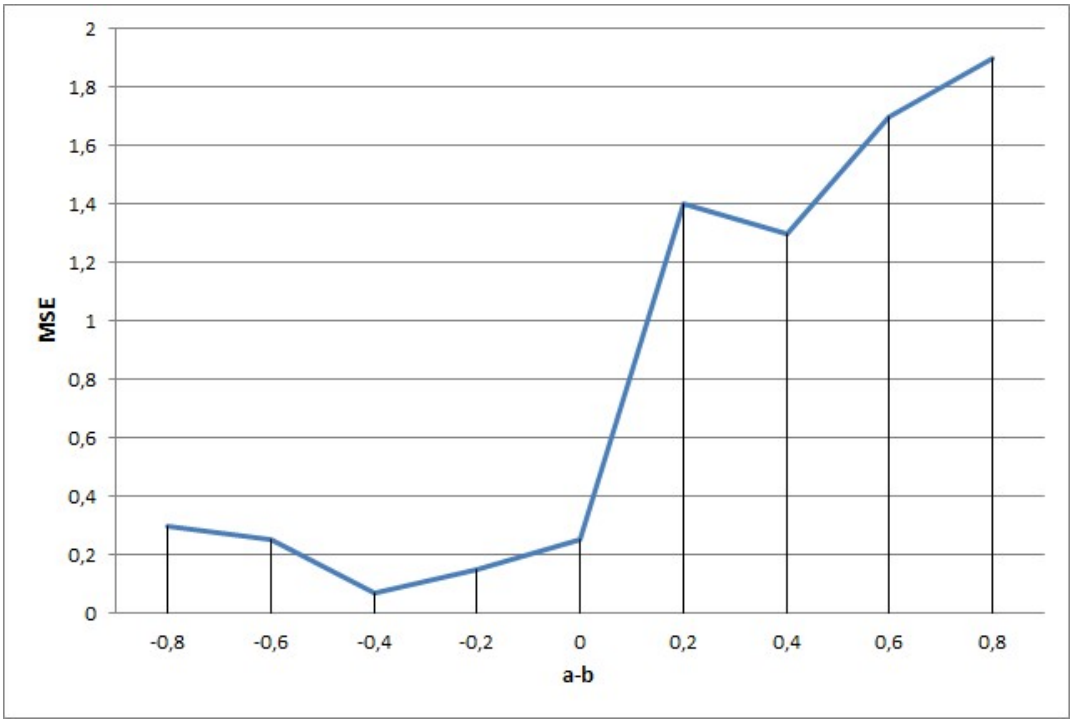
Όπως αναφέρθηκε παραπάνω , οι παράμετροι a, b του αλγορίθμου HGAPSO εκφράζουν στην ουσία το ποσοστό των συνολικών επαναλήψεων του αλγορίθμου που θα εκτελεστούν από τον ΓΑ από τον αλγόριθμο PSO αντίστοιχα . Για την εύρεση των βέλτιστων τιμών των παραμέτρων αυτών , έγινε χρήση του λογισμικού το οποίο περιγράφεται στο επόμενο κεφάλαιο . Ειδικότερα , για κάθε σύνολο εκπαίδευσης εκτελέστηκε ο αλγόριθμος HGAPSO για συγκεκριμένες τιμές στο μέγεθος του πληθυσμού (100) και στον αριθμό των επαναλήψεων (100) . Οι διαφορετικές τιμές στις παραμέτρους a, b που δοκιμάστηκαν παρουσιάζονται στον Πίνακα 6 που ακολουθεί .

# Πείραμα	a	b
1	0.1	0.9
2	0.2	0.8
3	0.3	0.7
4	0.4	0.6

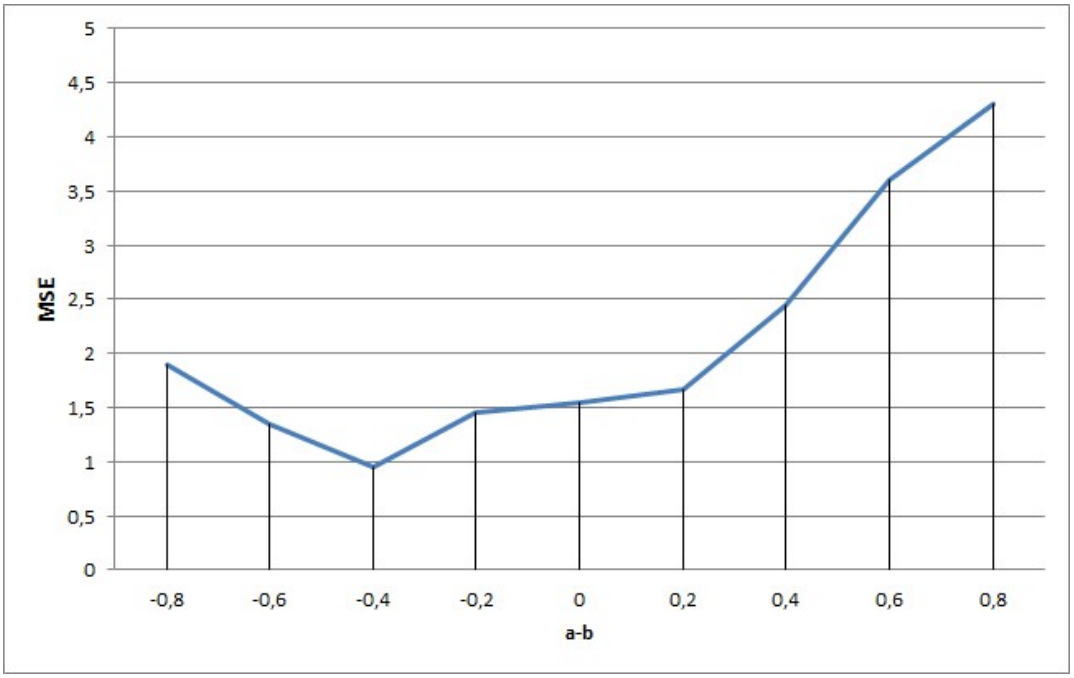
5	0.5	0.5
6	0.6	0.4
7	0.7	0.3
8	0.8	0.2
9	0.9	0.1

Πίνακας 6: τα πειράματα τα οποία εκτελέστηκαν για την απόδοση των βέλτιστων τιμών στις παραμέτρους a, b του αλγορίθμου HGAPSO .

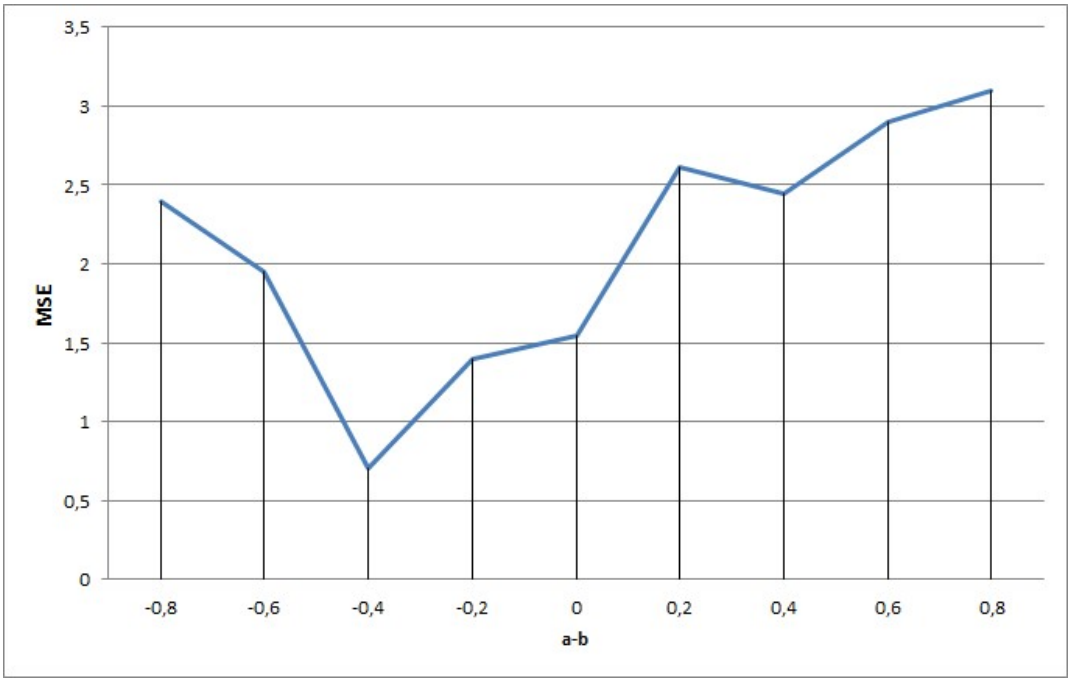
Επειδή ο αλγόριθμος είναι πιθανοτικός , το κάθε πείραμα εκτελέστηκε 10 φορές και υπολογίστηκε ο μέσος όρος των αποτελεσμάτων . Τα αποτελέσματα παρουσιάζονται στις εικόνες 22-26 . Ο οριζόντιος άξονας έχει τις διαφορές των παραμέτρων a-b ενώ ο κάθετος το MSE .



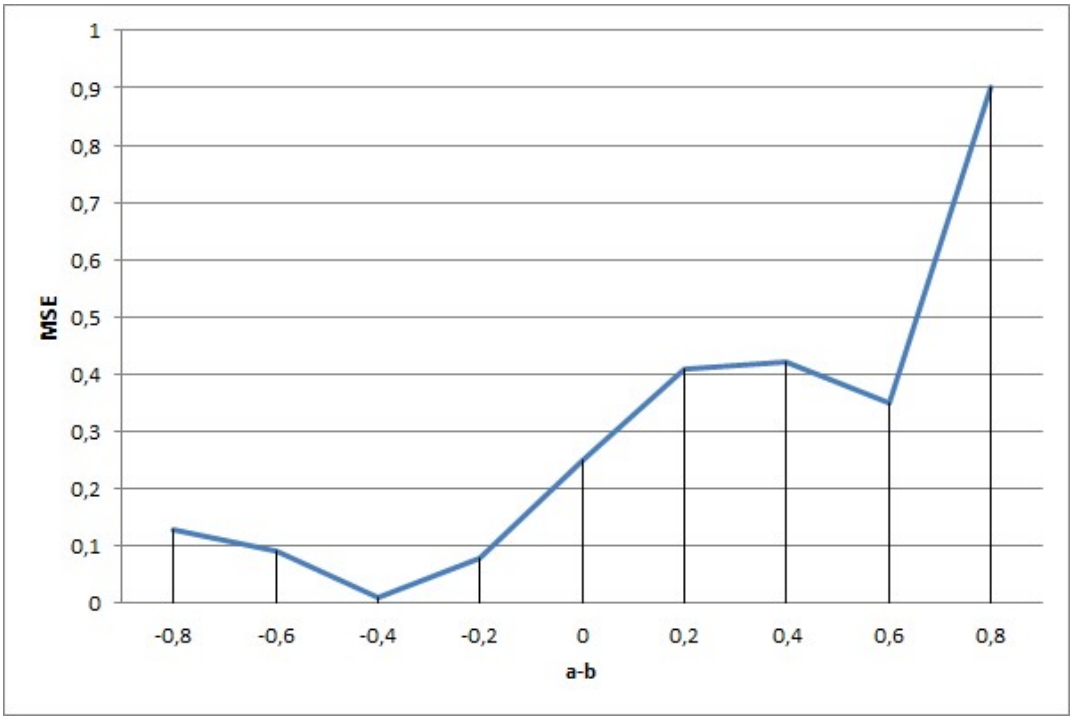
Εικόνα 22: το αποτέλεσμα της εκτέλεσης για το σύνολο εκπαίδευσης Abalone .



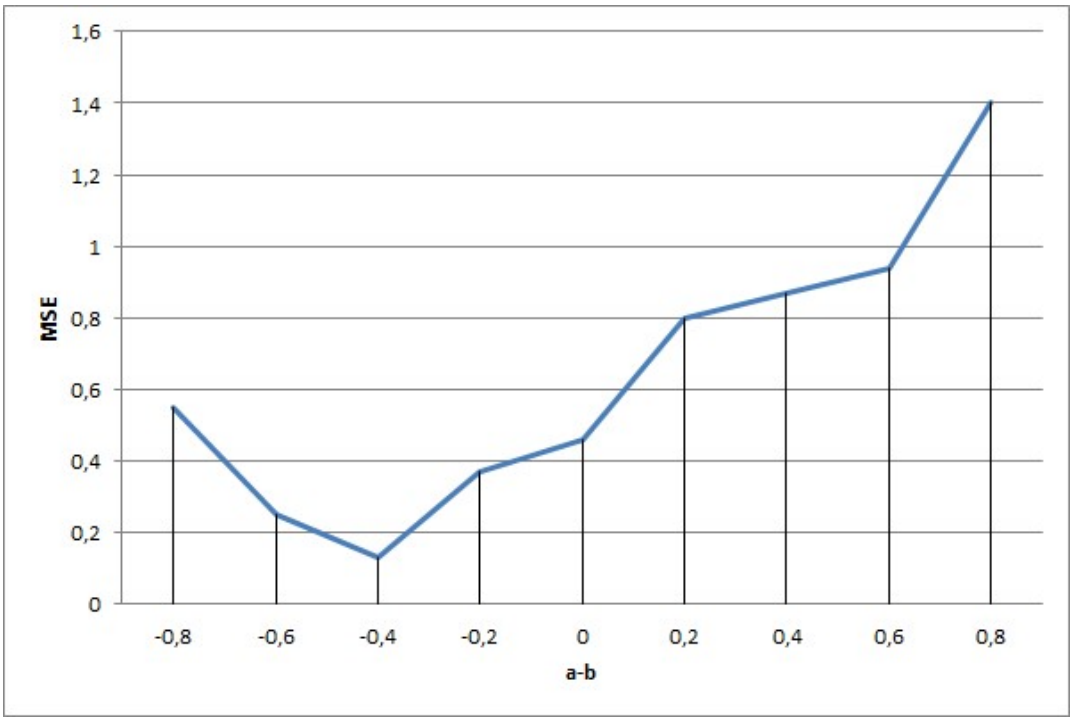
Εικόνα 23: το αποτέλεσμα της εκτέλεσης για το σύνολο εκπαίδευσης Balance-Scale .



Εικόνα 24: το αποτέλεσμα της εκτέλεσης για το σύνολο εκπαίδευσης Car .



Εικόνα 25: το αποτέλεσμα της εκτέλεσης για το σύνολο εκπαίδευσης Cloud .



Εικόνα 26: το αποτέλεσμα της εκτέλεσης για το σύνολο εκπαίδευσης Iris .

Από τα παραπάνω πειράματα προκύπτει ότι η βέλτιστη τιμή της διαφοράς $a - b$ είναι η -0.4 , η οποία αντιστοιχεί σε a = 0.3 και b = 0.7 . Αυτό πρακτικά σημαίνει ότι ο αλγόριθμος HGAPSO παρουσιάζει την βέλτιστη απόδοση όταν το 30% των συνολικών εκτελέσεων εκτελούνται από τον ΓΑ και το υπόλοιπο 70% από τον αλγόριθμο PSO .

6.5 Μεθοδολογία εκτέλεσης των αλγορίθμων και καταγραφής των πειραματικών αποτελεσμάτων

Η μεθοδολογία η οποία θα ακολουθήσουμε για την εκτέλεση των πειραμάτων παρουσιάζεται παρακάτω . Τα παρακάτω πειράματα θα εκτελεστούν για κάθε ένα από τα 5 σύνολα δεδομένων εκπαίδευσης . Οι εκτελέσεις των αλγορίθμων που θα πραγματοποιήσουμε είναι οι εξής :

1. Εκτέλεση του αλγορίθμου Οπισθοδιάδοσης του Λάθους για αριθμό εποχών από 100 μέχρι 1000 , με βήμα αύξησης το 100 .
2. Εκτέλεση του ΓΑ για pc = 0.5 , pm = 0.5 , για μέγεθος πληθυσμού ίσο με 100 και αριθμό επαναλήψεων από 10 μέχρι 100 με βήμα αύξησης 10 .
3. Εκτέλεση του αλγορίθμου PSO για $c1$ = 0.5 , $c2$ = 0.5 , για μέγεθος σμήνους ίσο με 100 και αριθμό επαναλήψεων από 10 μέχρι 100 με βήμα αύξησης 10 .
4. Εκτέλεση του αλγορίθμου HGAPSO για pc = 0.5 , pm = 0.5 , $c1$ = 0.5 , $c2$ = 0.5 , για μέγεθος πληθυσμού ίσο με 100 και αριθμό επαναλήψεων από 10 μέχρι 100 με βήμα αύξησης 10 .

Επειδή οι αλγόριθμοι είναι πιθανοτικοί , τα παραπάνω πειράματα θα εκτελεστούν 10 φορές και θα υπολογιστεί ο μέσος όρος των αποτελεσμάτων . Τα αποτελέσματα των πειραμάτων θα παρουσιαστούν ως εξής :

1. Παρουσίαση των αποτελεσμάτων ανά αλγόριθμο , έτσι ώστε να μελετηθεί η απόδοση του κάθε αλγορίθμου για τις διαφορετικές τιμές που θα πάρουν οι παράμετροί του .
2. Παρουσίαση των καλύτερων επιδόσεων των αλγορίθμων συγκεντρωτικά , έτσι ώστε να μπορέσουμε να δούμε ποιος εμφανίζει τα καλύτερα αποτελέσματα σε κάθε σύνολο εκπαίδευσης .

ΚΕΦΑΛΑΙΟ 7: ΛΟΓΙΣΜΙΚΟ ΕΚΤΕΛΕΣΗΣ ΠΕΙΡΑΜΑΤΩΝ

7.1 Εισαγωγή

Στο κεφάλαιο αυτό παρουσιάζεται το λογισμικό το οποίο υλοποιήθηκε για την σύγκριση των 4 αλγορίθμων εκπαίδευσης (Οπισθοδιάδοσης του λάθους, ΓΑ, PSO και HGAPSO) . Το λογισμικό αυτό υλοποιεί και τους 4 αλγορίθμους εκπαίδευσης και είναι παραμετρικό όσον αφορά τα παρακάτω μεγέθη :

- Τα δεδομένα εκπαίδευσης .
- Τον αριθμό των επαναλήψεων που θα εκτελεστεί ο εκάστοτε αλγόριθμος .
- Το μέγεθος του πληθυσμού που θα εξελίξει ο εκάστοτε εξελικτικός αλγόριθμος .
- Τις διάφορες παραμέτρους του εκάστοτε αλγορίθμου .

Παρακάτω στο κεφάλαιο αυτό δίνονται με συνοπτικό τρόπο οι λεπτομέρειες υλοποίησης του κάθε αλγορίθμου και στην συνέχεια τα κριτήρια σύγκρισης που θα χρησιμοποιηθούν για την ποσοτικοποίηση της απόδοσής του .

7.2 Γενικές πληροφορίες

Για την υλοποίηση του λογισμικού έγινε χρήση της γλώσσας προγραμματισμού Java και του περιβάλλοντος NetBeans . Όλοι οι αλγόριθμοι υλοποιήθηκαν μέσα στα όρια του ίδιου λογισμικού . Έγινε προσπάθεια έτσι ώστε η υλοποίηση του κάθε εξελικτικού αλγορίθμου να γίνει με ομοιόμορφο τρόπο έτσι ώστε αυτό το οποίο να διαφέρει σε κάθε αλγόριθμο να είναι ο τρόπος κίνησης των ατόμων του πληθυσμού μέσα στον χώρο αναζήτησης και όχι ο τρόπος αναπαράστασης και μοντελοποίησης του προβλήματος .

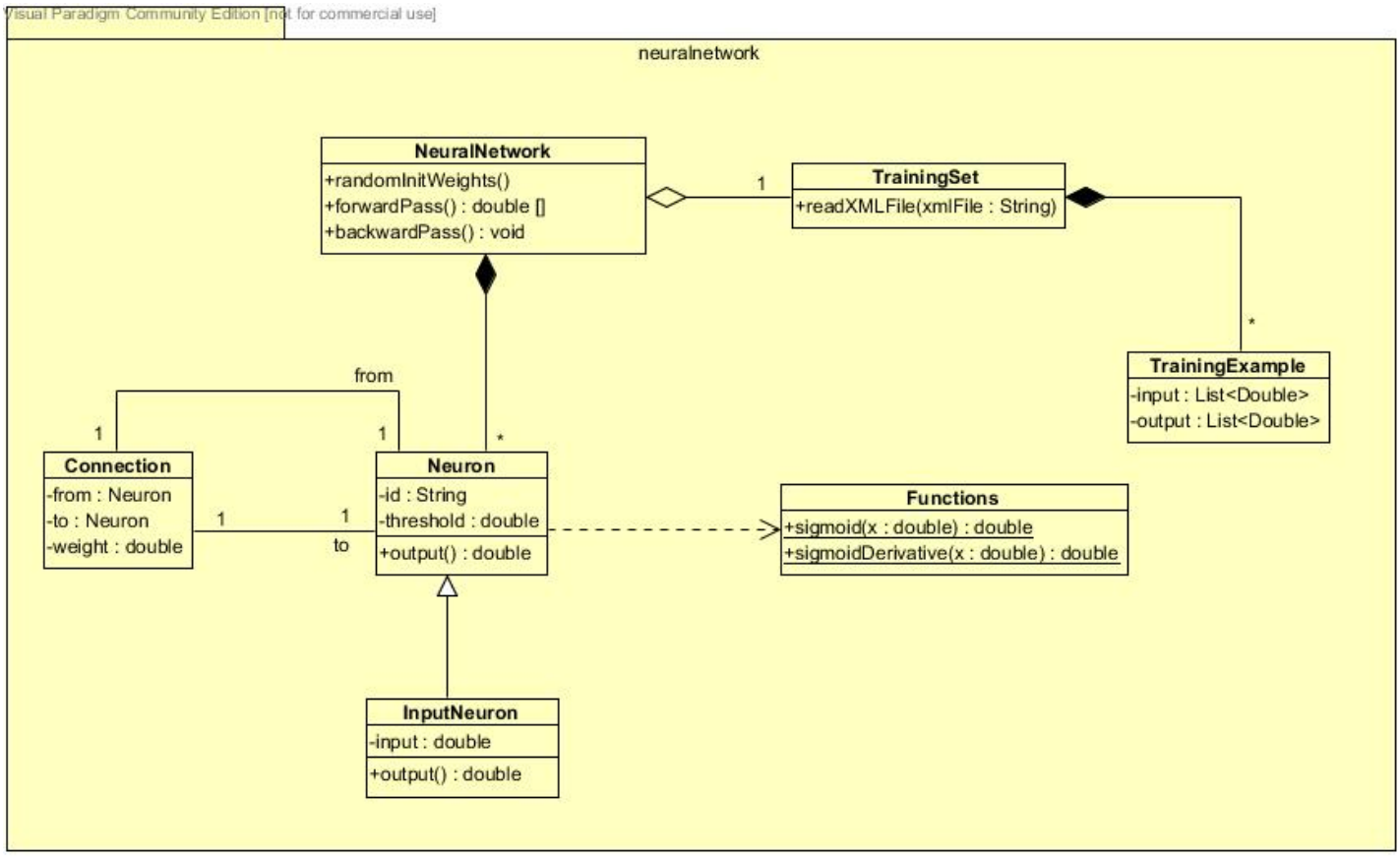
7.3 Λεπτομέρειες υλοποίησης του λογισμικού

7.3.1 Υλοποίηση του ΤΝΔ

Όλες οι απαραίτητες κλάσεις για την υλοποίηση ενός ΤΝΔ βρίσκονται μέσα στο πακέτο neuralnetwork . Ο αρχιτεκτονικός σχεδιασμός του πακέτου neuralnetwork παρουσιάζεται στην εικόνα 27 της επόμενης σελίδας . Στο πακέτο αυτό έχουν υλοποιηθεί οι παρακάτω κλάσεις :

- **Functions** : περιέχει την σιγμοειδή συνάρτηση και την παράγωγό της .
- **Neuron** : αντιστοιχεί σε έναν νευρώνα του ΤΝΔ . Κάθε νευρώνας χρησιμοποιεί ως συνάρτηση ενεργοποίησης την σιγμοειδή συνάρτηση και διαθέτει ένα σύνολο από συνδέσεις εισόδου και εξόδου . Η κλάση έχει την μέθοδο *output()* , η οποία παράγει την έξοδο του νευρώνα .
- **InputNeuron** : αντιστοιχεί σε έναν νευρώνα εισόδου , ο οποίος επιπρόσθετα αποθηκεύει την τιμή εισόδου . Η κλάση επανορίζει την μέθοδο *output()* , η οποία απλά επιστρέφει την τιμή εισόδου .
- **Connection** : αντιστοιχεί σε μία σύνδεση μεταξύ δύο νευρώνων . Η κλάση αποθηκεύει το βάρος της σύνδεσης ως πεδίο .
- **TrainingExample** : αντιστοιχεί σε ένα παράδειγμα εισόδου / εξόδου .
- **TrainingSet** : αντιστοιχεί σε ένα σύνολο εκπαίδευσης , το οποίο αποτελείται από παραδείγματα εισόδου / εξόδου . Η κλάση έχει την μέθοδο *readXMLFile()* από την οποία μπορεί να διαβάσει τα δεδομένα από ένα αρχείο XML .
- **NeuralNetwork** : αντιστοιχεί σε ένα ΤΝΔ . Η κλάση έχει την μέθοδο *randomInitWeights()* , η οποία αρχικοποιεί με τυχαίο τρόπο τα βάρη του ΤΝΔ , την μέθοδο *fordwardPass()* η οποία πραγματοποιεί το εμπρόσθιο πέρασμα και επιστρέφει

την έξοδο του ΤΝΔ αλλά και την μέθοδο *backwardPass()* η οποία πραγματοποιεί το πέρασμα προς τα πίσω .

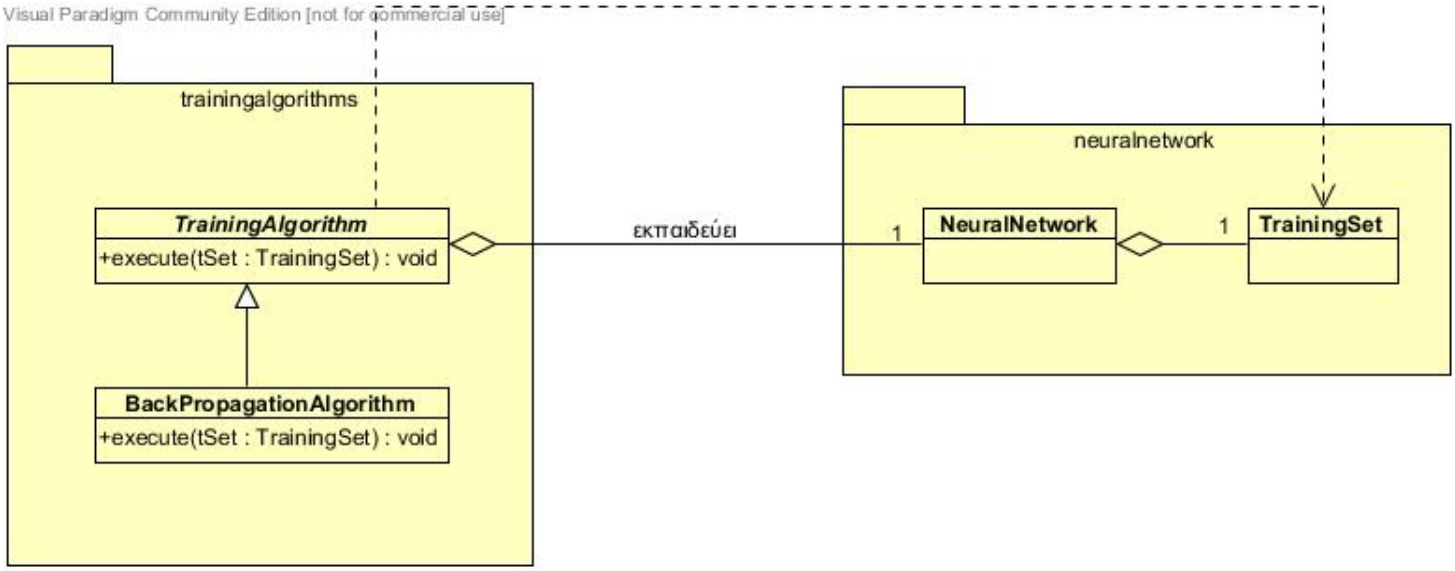


Εικόνα 27: αρχιτεκτονικός σχεδιασμός του πακέτου neuralnetwork .

7.3.2 Υλοποίηση του Αλγορίθμου Οπισθοδιάδοσης του Λάθους

Όλες οι απαραίτητες κλάσεις για την υλοποίηση του Αλγορίθμου Οπισθοδιάδοσης του Λάθους βρίσκονται μέσα στο πακέτο trainingalgorithms . Ο αρχιτεκτονικός σχεδιασμός του πακέτου trainingalgorithms παρουσιάζεται στην εικόνα 28 της επόμενης σελίδας . Στο πακέτο αυτό έχουν υλοποιηθεί οι παρακάτω κλάσεις :

- **TrainingAlgorithm** : αντιστοιχεί σε έναν οποιοδήποτε συμβατικό (μη εξελικτικό) αλγόριθμο εκπαίδευσης ΤΝΔ . Η κλάση αποθηκεύει ως πεδίο το ΤΝΔ το οποίο πρόκειται να εκπαιδεύσει και για να το πετύχει αυτό χρησιμοποιεί αναγκαστικά δεδομένα εκπαίδευσης (TrainingSet) . Η κλάση είναι αφηρημένη και κάθε υποκλάση που την υλοποιεί θα πρέπει να υλοποιεί την μέθοδο *execute()* .
- **BackPropagationAlgorithm** : αντιστοιχεί στον αλγόριθμο Οπισθοδιάδοσης του Λάθους . Ειδικότερα , αποτελεί υποκλάση της TrainingAlgorithm και υλοποιεί την μέθοδο *execute()* , με όλα τα απαραίτητα βήματα για την υλοποίηση του αλγορίθμου .



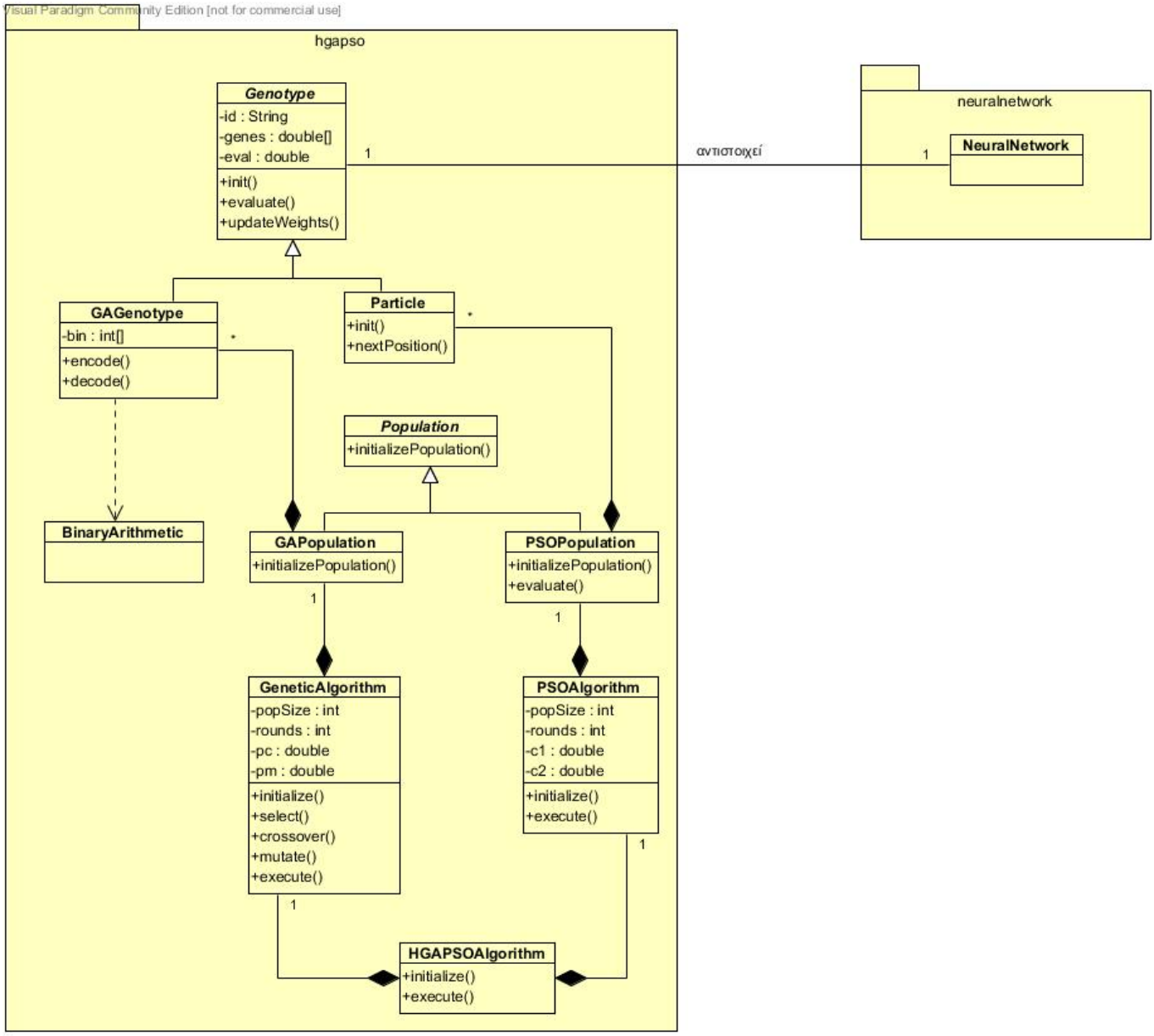
Εικόνα 28: αρχιτεκτονικός σχεδιασμός του πακέτου trainingalgorithms , το οποίο χρησιμοποιεί κλάσεις από το πακέτο neuralnetwork .

7.3.3 Υλοποίηση των Εξελικτικών Αλγορίθμων

Όλες οι απαραίτητες κλάσεις για την υλοποίηση των Εξελικτικών Αλγορίθμων βρίσκονται μέσα στο πακέτο hgapso . Ο αρχιτεκτονικός σχεδιασμός του πακέτου hgapso παρουσιάζεται στην εικόνα 29 της επόμενης σελίδας . Στο πακέτο αυτό έχουν υλοποιηθεί οι παρακάτω κλάσεις :

- **Genotype :** πρόκειται για μία αφηρημένη κλάση , η οποία αντιστοιχεί σε ένα οποιοδήποτε άτομο ενός πληθυσμού κάποιου εξελικτικού αλγορίθμου . Ο κάθε γονότυπος αντιστοιχεί στην ουσία σε ένα διάνυσμα πραγματικών τιμών , όπου στην περίπτωσή μας είναι τα βάρη ενός ΤΝΔ . Το ΤΝΔ το οποίο ο γονότυπος εκφράζει αποθηκεύεται σε αυτόν ως πεδίο , κάτι που δικαιολογεί την σύνδεση με την κλάση NeuralNetwork . Οι μέθοδοι της κλάσης Genotype είναι η *init()* , η οποία αρχικοποιεί τα γονίδια του ατόμου , η evaluate() , η οποία αξιολογεί το άτομο και η *updateWeights()* , η οποία αντιγράφει τα γονίδια του γονοτύπου στο ΤΝΔ .
- **GAGenotype :** πρόκειται για μία υποκλάση της Genotype , η οποία αντιστοιχεί σε ένα άτομο του πληθυσμού του ΓΑ . Η κλάση περιλαμβάνει επιπλέον τις μεθόδους *encode()* και *decode()* , για την κωδικοποίηση και την αποκωδικοποίηση αντίστοιχα του γονοτύπου σε / από μια δυαδική συμβολοσειρά . Η κλάση κάνει χρήση των μεθόδων της κλάσης BinaryArithmetic για τον σκοπό αυτό .
- **Particle :** πρόκειται για μία υποκλάση της Genotype , η οποία αντιστοιχεί σε ένα μόριο του σμήνους του αλγορίθμου PSO . Η κλάση περιλαμβάνει επιπλέον τις μεθόδους *init()* και *nextPosition()* , για την αρχικοποίηση του ατόμου καθώς και την μετακίνησή του στην επόμενη θέση στον χώρο αναζήτησης .
- **BinaryArithmetic :** βοηθητική κλάση , η οποία περιλαμβάνει μεθόδους για την κωδικοποίηση και την αποκωδικοποίηση πραγματικών διανυσμάτων σε / από δυαδικές συμβολοσειρές .
- **Population :** πρόκειται για μία αφηρημένη κλάση , η οποία αντιστοιχεί στον πληθυσμό ενός εξελικτικού αλγορίθμου . Περιέχει την αφηρημένη μέθοδο *initializePopulation()* , την οποία οι υποκλάσεις της θα πρέπει υποχρεωτικά να επανορίσουν .
- **GAPopulation :** πρόκειται για μία υποκλάση της Population , η οποία αντιστοιχεί στον πληθυσμό του ΓΑ .
- **PSOPopulation :** πρόκειται για μία υποκλάση της Population , η οποία αντιστοιχεί στον πληθυσμό του αλγορίθμου PSO .

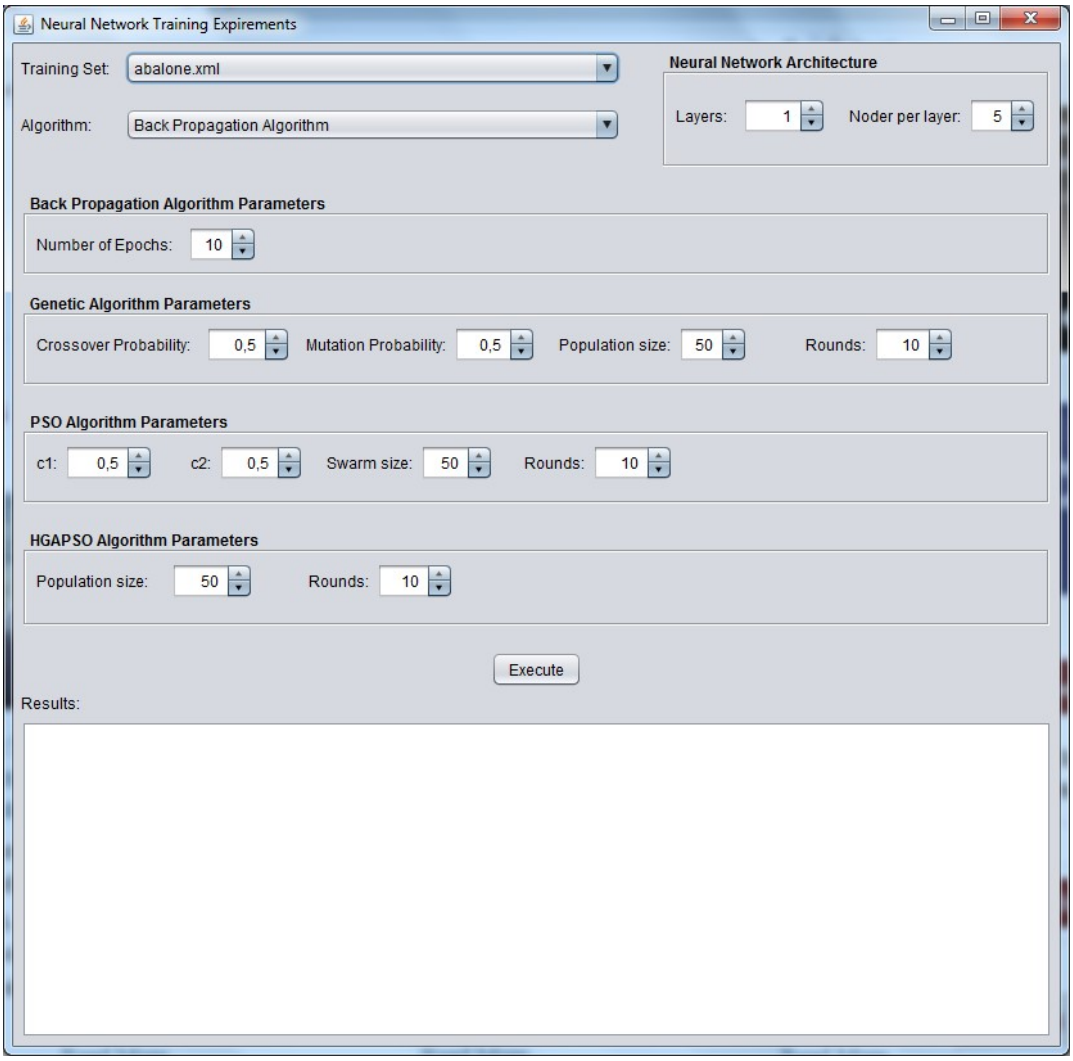
- **GeneticAlgorithm :** η κλάση αυτή υλοποιεί τον ΓΑ . Αποθηκεύει ως πεδία τις παραμέτρους του ΓΑ και υλοποιεί μεθόδους για την αρχικοποίηση του αλγορίθμου καθώς και τις διαδικασίες της επιλογής , διασταύρωσης και μετάλλαξης .
- **PSOAlgorithm :** η κλάση αυτή υλοποιεί τον αλγόριθμο PSO . Αποθηκεύει ως πεδία τις παραμέτρους του αλγορίθμου PSO και έχει μεθόδους για την αρχικοποίηση του αλγορίθμου καθώς και για την εκτέλεση της διαδικασίας εξέλιξης .
- **HGAPSOAlgorithm :** η κλάση αυτή υλοποιεί τον αλγόριθμο HGAPSO . Στην ουσία αποτελείται από ένα αντικείμενο της κλάσης GeneticAlgorithm και ένα αντικείμενο της κλάσης PSOAlgorithm .



Εικόνα 29: αρχιτεκτονικός σχεδιασμός του πακέτου hgapso , το οποίο χρησιμοποιεί την κλάση NeuralNetwork από το πακέτο neuralnetwork .

7.4 Εγχειρίδιο χρήσης του λογισμικού

Η κεντρική οθόνη του λογισμικού HGAPSO παρουσιάζεται στην εικόνα 30 :



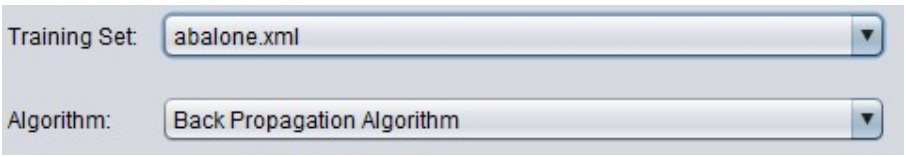
Εικόνα 30: η κεντρική οθόνη του λογισμικού

Από την παραπάνω εικόνα μπορούμε να δούμε ότι η κεντρική οθόνη του λογισμικού απαρτίζεται από τις εξής 7 περιοχές :

1. Τμήμα επιλογής συνόλου εκπαίδευσης και αλγορίθμου εκπαίδευσης
2. Τμήμα επιλογής της τοπολογίας του ΤΝΔ
3. Τμήμα των παραμέτρων του αλγορίθμου BP
4. Τμήμα των παραμέτρων του ΓΑ
5. Τμήμα των παραμέτρων του αλγορίθμου PSO
6. Τμήμα των παραμέτρων του αλγορίθμου HGAPSO
7. Περιοχή καταγραφής των αποτελεσμάτων εκτέλεσης

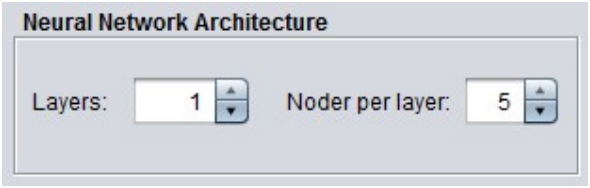
Κάθε ένα από τα παραπάνω τμήματα παρουσιάζεται παρακάτω στις εικόνες 31 έως 37 . Σε κάθε εικόνα παρέχεται μία σύντομη περιγραφή της λειτουργικότητας του κάθε τμήματος .

1. Τμήμα επιλογής του συνόλου εκπαίδευσης και του αλγορίθμου εκπαίδευσης



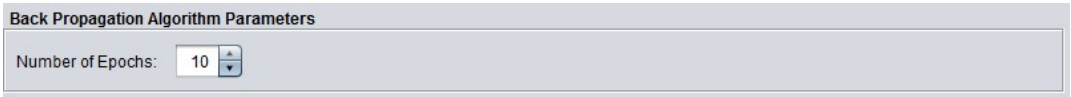
Εικόνα 31: τμήμα επιλογής του συνόλου εκπαίδευσης και του αλγορίθμου εκπαίδευσης . Τα σύνολα εκπαίδευσης είναι στην ουσία αρχεία XML (η δομή τους περιγράφεται παρακάτω) και πρέπει να βρίσκονται στον κατάλογο trainingsets , ο οποίος θα πρέπει να βρίσκεται στον ίδιο τρέχον κατάλογο με το αρχείο της εφαρμογής .

2. Τμήμα επιλογής της τοπολογίας του ΤΝΔ



Εικόνα 32: τμήμα επιλογής της τοπολογίας του ΤΝΔ . Η πρώτη επιλογή αφορά τον αριθμό των επιπέδων και η δεύτερη τον αριθμό των νευρώνων ανά επίπεδο . Θα πρέπει να σημειωθεί ότι το κάθε ΤΝΔ που δημιουργείται είναι εμπρόσθιας τροφοδότησης και πλήρως διασυνδεδεμένο . Ειδικότερα , όλοι οι νευρώνες κάθε επιπέδου (πλην του επιπέδου εξόδου) δίνουν έξοδο σε κάθε νευρώνα του επόμενου επιπέδου .

3. Τμήμα των παραμέτρων του αλγορίθμου BP

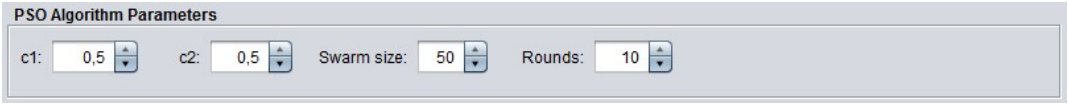


Εικόνα 33: τμήμα επιλογής των παραμέτρων του αλγορίθμου BP . Η μόνη παράμετρος η οποία μπορούμε να θέσουμε είναι ο αριθμός των εποχών (epochs) για τις οποίες ο αλγόριθμος θα εκτελεστεί .

4. Τμήμα των παραμέτρων του ΓΑ

Εικόνα 34: τμήμα επιλογής των παραμέτρων του ΓΑ . Οι παράμετροι στις οποίες ο χρήστης μπορεί να θέσει τιμές είναι η πιθανότητα διασταύρωσης , η πιθανότητα μετάλλαξης, το μέγεθος του πληθυσμού και ο αριθμός των κύκλων εξέλιξης .

5. Τμήμα των παραμέτρων του αλγορίθμου PSO



Εικόνα 35: τμήμα επιλογής των παραμέτρων του αλγορίθμου PSO . Οι παράμετροι στις οποίες ο χρήστης μπορεί να θέσει τιμές είναι οι παράμετροι c_1 και c_2 του αλγορίθμου , το μέγεθος του σμήνους και ο αριθμός των κύκλων εκπαίδευσης .

6. Τμήμα των παραμέτρων του αλγορίθμου HGAPSO

Εικόνα 36: τμήμα επιλογής των παραμέτρων του αλγορίθμου HGAPSO . Οι παράμετροι στις οποίες ο χρήστης μπορεί να θέσει τιμές είναι το μέγεθος του πληθυσμού και ο αριθμός των κύκλων εκπαίδευσης . Οι τιμές στις υπόλοιπες παραμέτρους , καθώς ο αλγόριθμος HGAPSO αποτελείται τόσο από τον ΓΑ όσο και από τον αλγόριθμο PSO ορίζονται στα αντίστοιχα τμήματα των αλγορίθμων αυτών .

7. Περιοχή καταγραφής των αποτελεσμάτων εκτέλεσης

Εικόνα 37: στο τμήμα αυτό καταγράφεται το αποτέλεσμα της εκτέλεσης του κάθε αλγορίθμου .

7.5 Δομή των αρχείων εκπαίδευσης

Τα αρχεία εκπαίδευσης είναι αρχεία XML , τα οποία αποθηκεύουν τα δεδομένα εκπαίδευσης . Η δομή ενός αρχείου εκπαίδευσης είναι η εξής :

```
<?xml version="1.0"?>
<trainingSet>
    <example>
        <inputs>
            <input>τιμή εισόδου</input>
            <input>τιμή εισόδου</input>
            ...
```

```
        </inputs>
        <outputs>
            <output>τιμή εξόδου</output>
            <output>τιμή εξόδου</output>
            …
        </outputs>
    </example>
    …
</trainingSet>
```

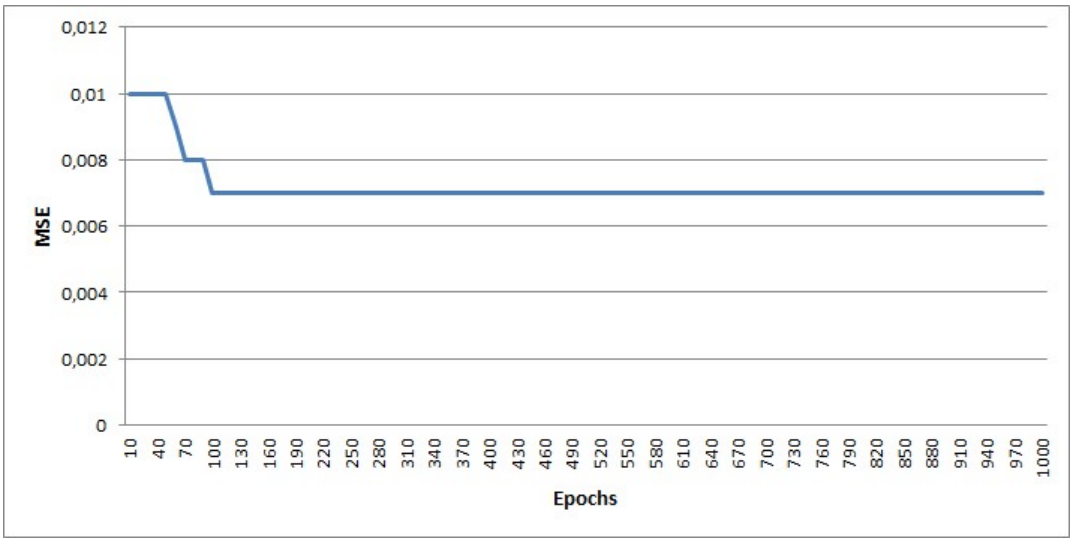
ΚΕΦΑΛΑΙΟ 8: ΠΕΙΡΑΜΑΤΙΚΑ ΑΠΟΤΕΛΕΣΜΑΤΑ

8.1 Εισαγωγή

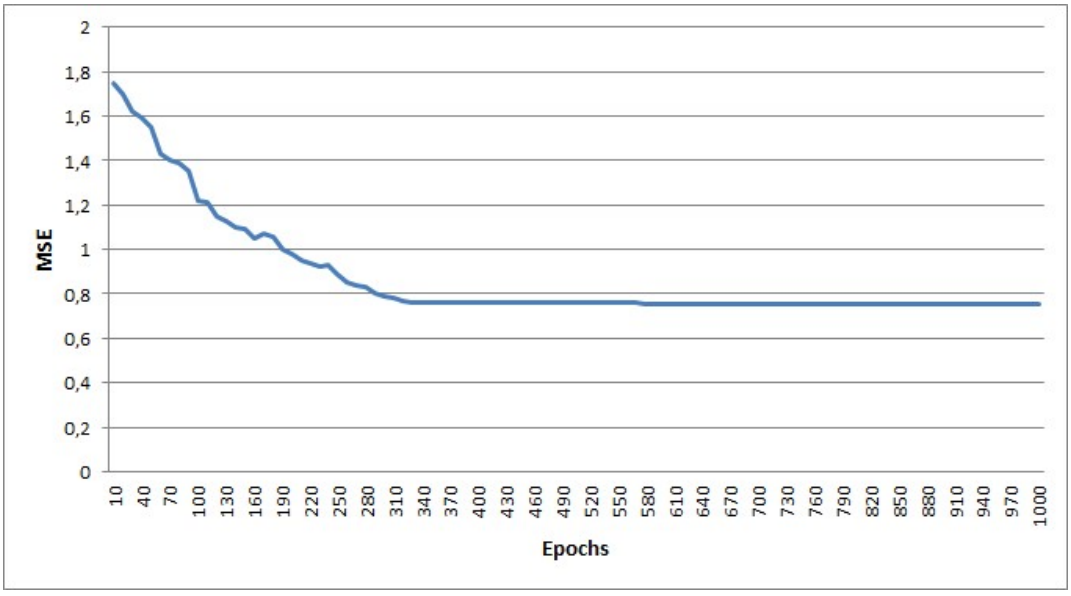
Στο κεφάλαιο αυτό παρουσιάζονται τα πειραματικά αποτελέσματα για κάθε έναν από τους 4 αλγορίθμους . Κριτήριο σύγκρισης για κάθε αλγόριθμο αποτελεί η συνάρτηση Μέσου Τετραγωνικού Σφάλματος (Mean Square Error – MSE) .

8.2 Πειραματικά αποτελέσματα του αλγορίθμου Οπισθοδιάδοσης του Λάθους

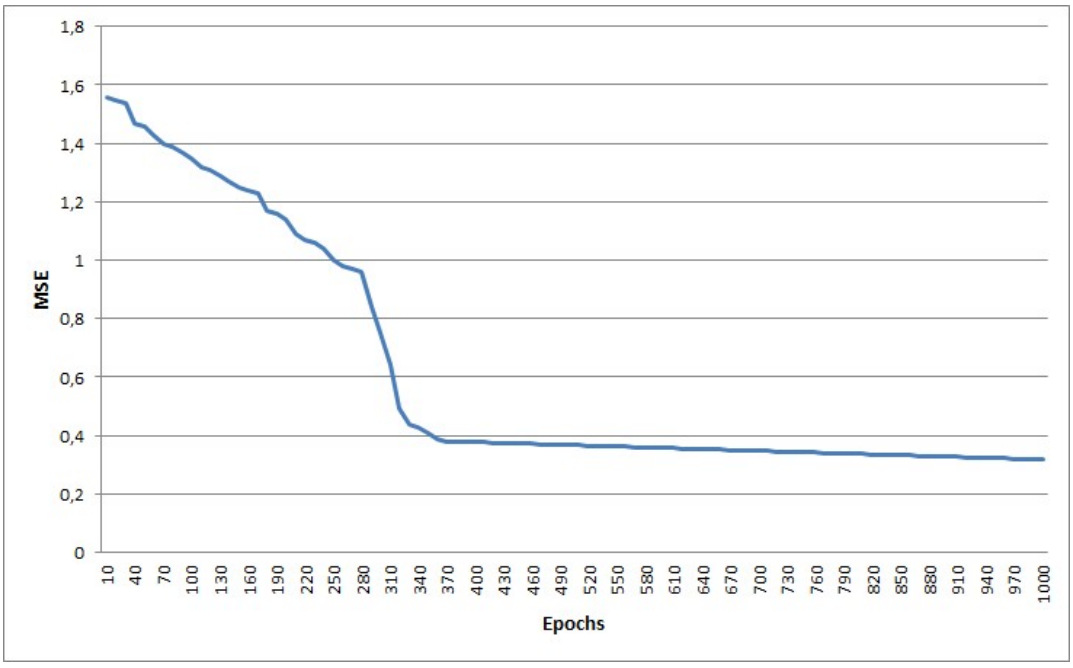
Τα πειραματικά αποτελέσματα για τον αλγόριθμο της Οπισθοδιάδοσης του Λάθους παρουσιάζονται στις Εικόνες 38 – 42 .



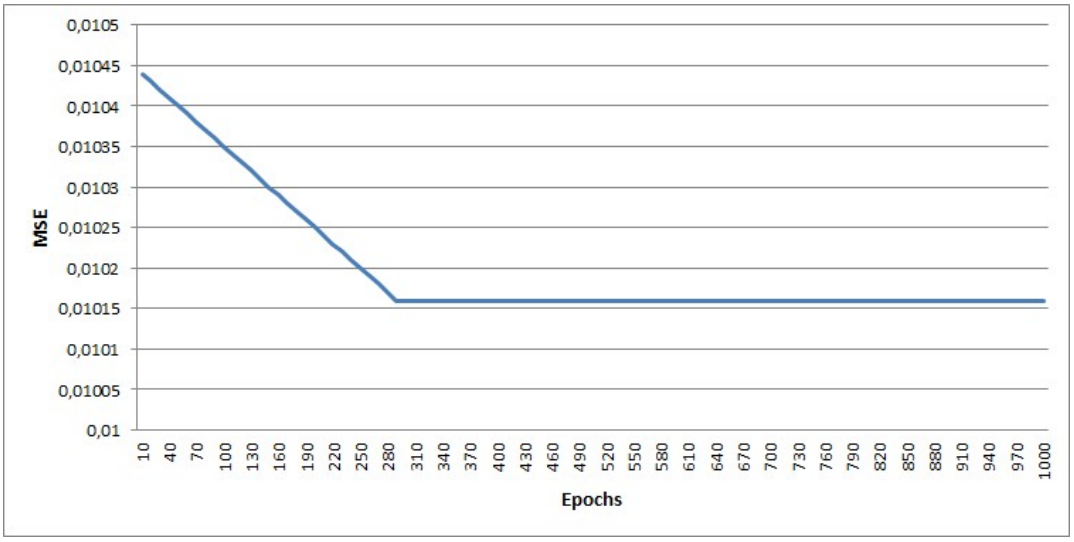
Εικόνα 38: πειραματικά αποτελέσματα του αλγορίθμου Οπισθοδιάδοσης του Λάθους (BP) για το σύνολο δεδομένων Abalone . Παρατηρούμε ότι ο αλγόριθμος BP αρχίζει να συγκλίνει προς την χαμηλότερη τιμή του MSE μετά από περίπου 100 εποχές . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.007 .



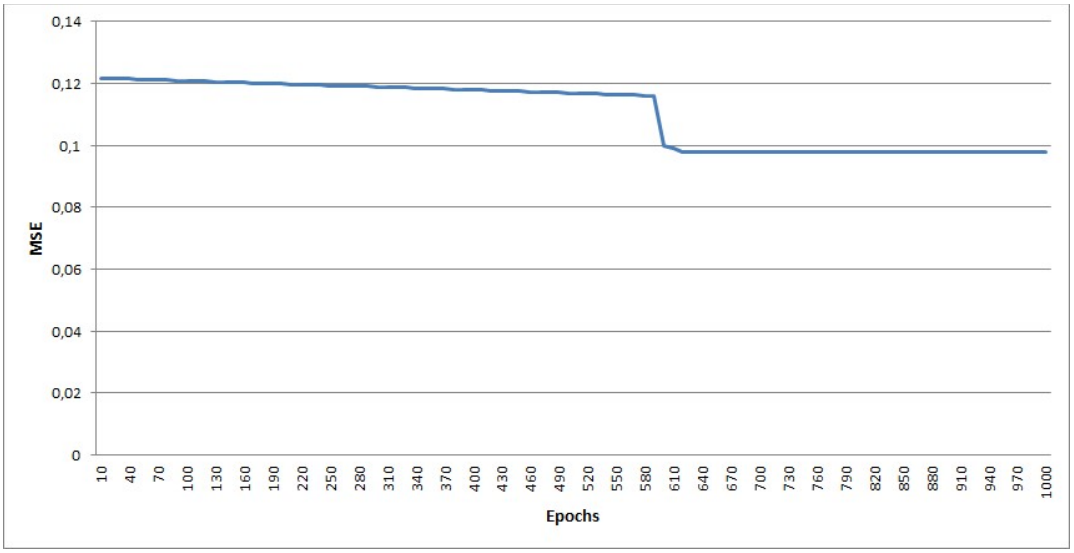
Εικόνα 39: πειραματικά αποτελέσματα του αλγορίθμου Οπισθοδιάδοσης του Λάθους (BP) για το σύνολο δεδομένων Balance-Scale . Παρατηρούμε ότι ο αλγόριθμος BP αρχίζει να συγκλίνει προς την χαμηλότερη τιμή του MSE μετά από περίπου 310 εποχές . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.75 .



Εικόνα 40: πειραματικά αποτελέσματα του αλγορίθμου Οπισθοδιάδοσης του Λάθους (BP) για το σύνολο δεδομένων Car . Παρατηρούμε ότι ο αλγόριθμος BP αρχίζει να συγκλίνει προς την χαμηλότερη τιμή του MSE μετά από περίπου 370 εποχές . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.31 .



Εικόνα 41: πειραματικά αποτελέσματα του αλγορίθμου Οπισθοδιάδοσης του Λάθους (BP) για το σύνολο δεδομένων Cloud . Παρατηρούμε ότι ο αλγόριθμος BP αρχίζει να συγκλίνει προς την χαμηλότερη τιμή του MSE μετά από περίπου 300 εποχές . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.01015 .



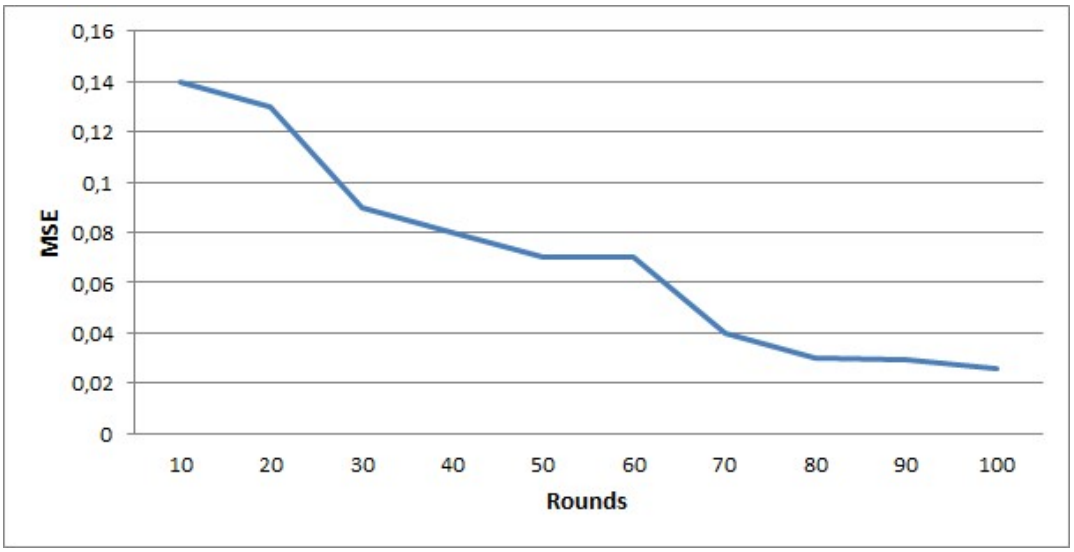
Εικόνα 42: πειραματικά αποτελέσματα του αλγορίθμου Οπισθοδιάδοσης του Λάθους (BP) για το σύνολο δεδομένων Iris . Παρατηρούμε ότι ο αλγόριθμος BP αρχίζει να συγκλίνει προς την χαμηλότερη τιμή του MSE μετά από περίπου 610 εποχές . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.09 .

Σχολιασμός των πειραματικών αποτελεσμάτων

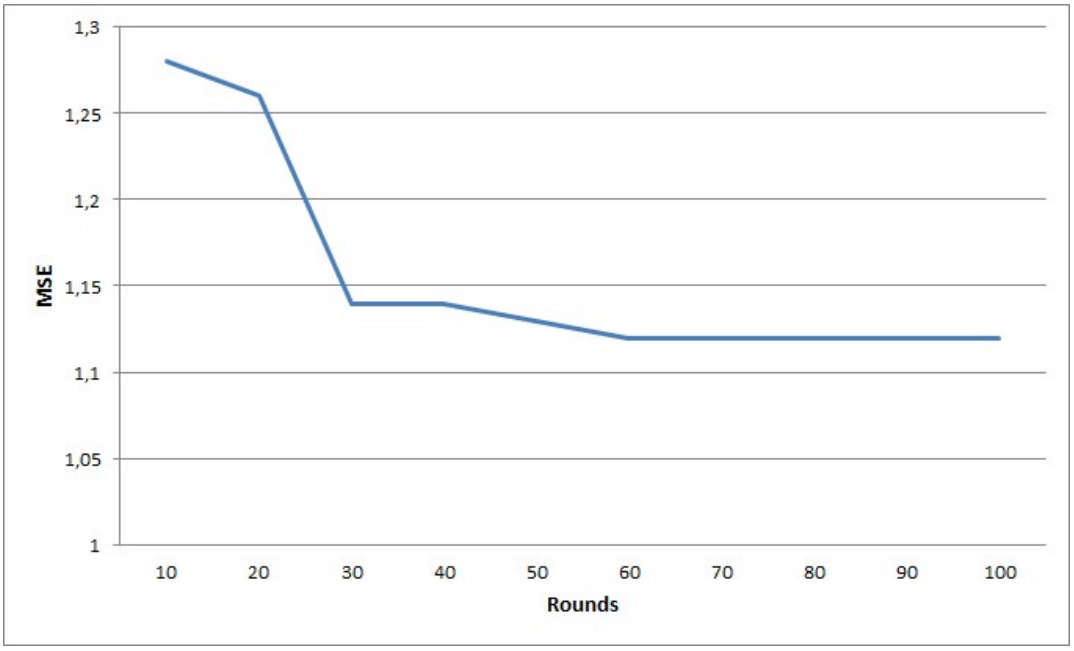
Ο αλγόριθμος BP εμφανίζει αρκετά καλά αποτελέσματα . Το κύριο μειονέκτημά του είναι ότι παγιδεύεται πάντα σε κάποιο τοπικό ελάχιστο μετά από έναν αριθμό εποχών και δεν μπορεί να ξεφύγει εύκολα από αυτό . Ακόμη, σε ορισμένα σύνολα δεδομένων εκπαίδευσης (π.χ. Iris) ο αλγόριθμος χρειάζεται ένα αρκετά μεγάλο αριθμό εποχών για να αρχίσει να συγκλίνει προς την μικρότερη τιμή του MSE .

8.3 Πειραματικά αποτελέσματα του Γενετικού Αλγορίθμου

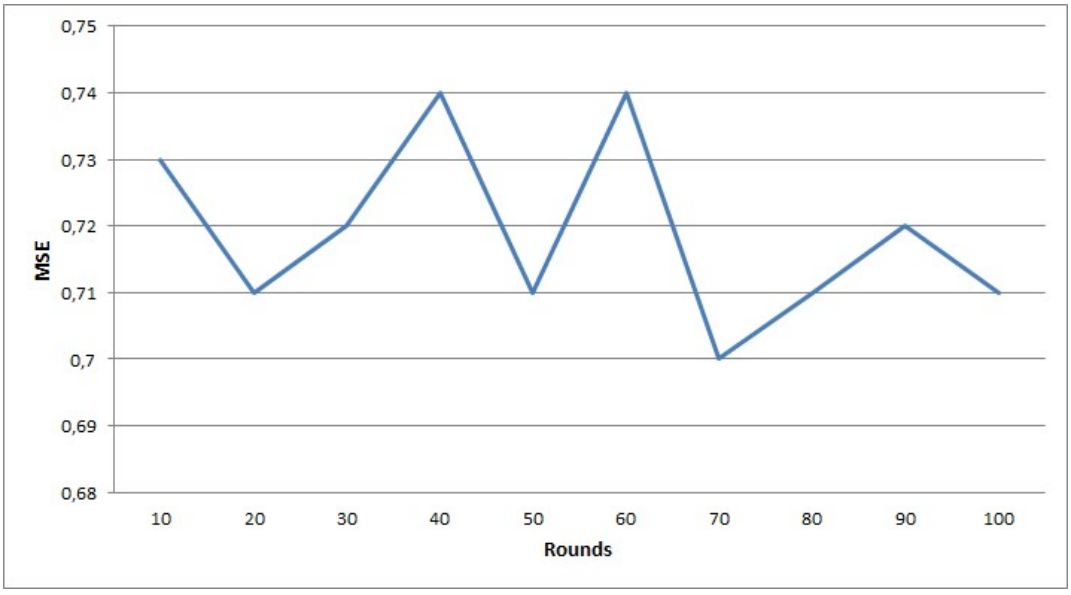
Τα πειραματικά αποτελέσματα για τον Γενετικό Αλγόριθμο παρουσιάζονται στις Εικόνες 43 – 47 .



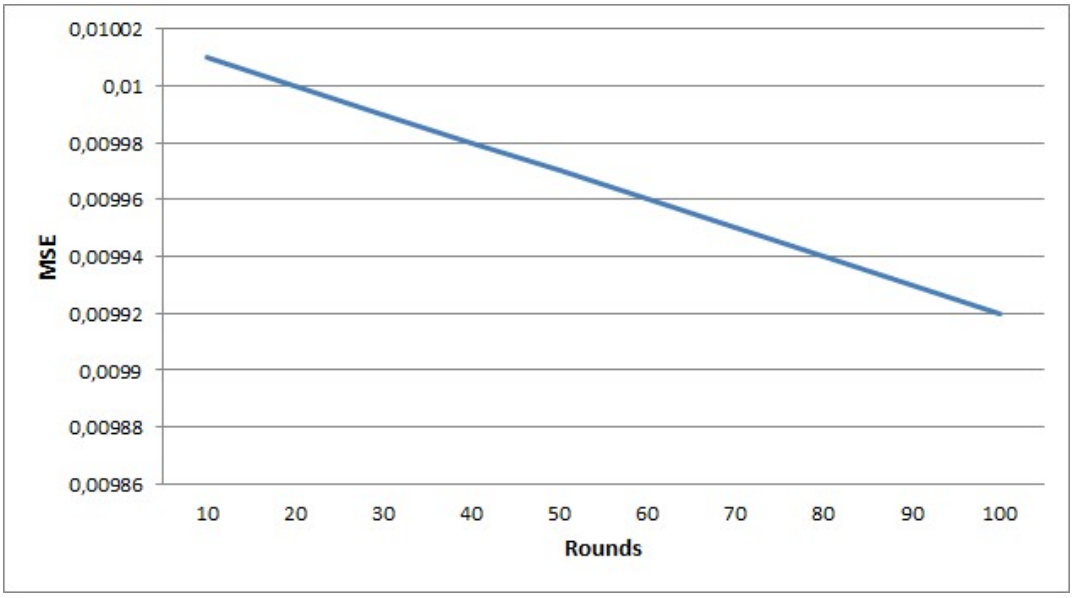
Εικόνα 43: πειραματικά αποτελέσματα του Γενετικού Αλγορίθμου (ΓΑ) για το σύνολο δεδομένων Abalone . Παρατηρούμε ότι ο ΓΑ παρουσιάζει μία φθίνουσα πορεία όσον αφορά την τιμή του MSE αναλογικά με την αύξηση του αριθμού των επαναλήψεων . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.026 .



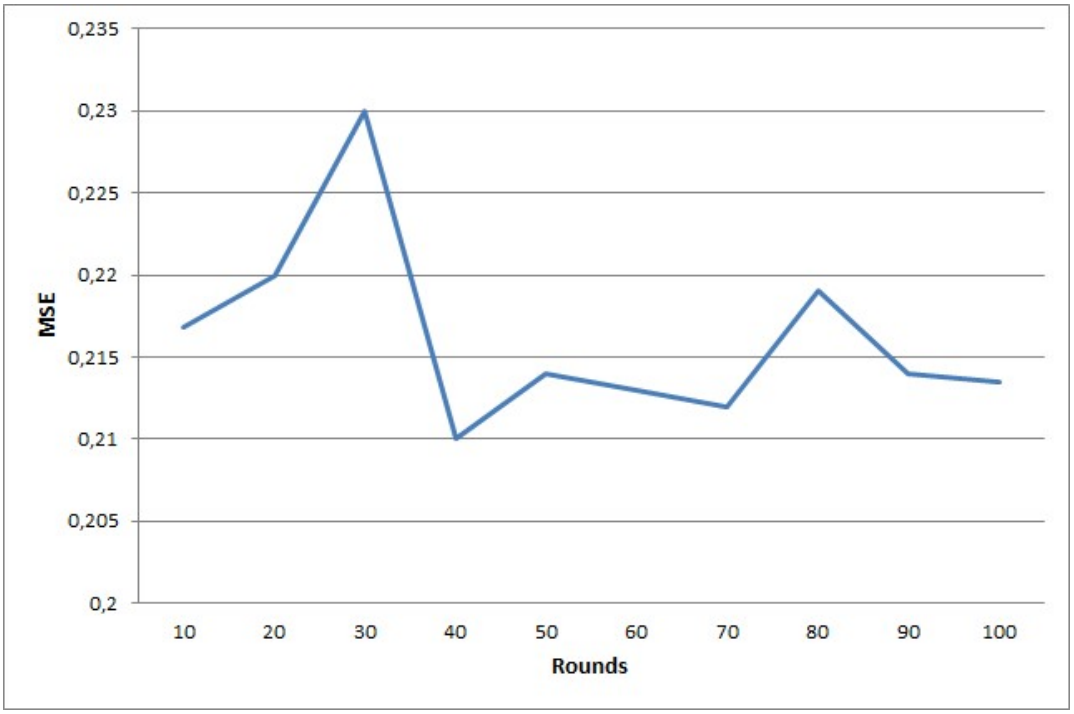
Εικόνα 44: πειραματικά αποτελέσματα του Γενετικού Αλγορίθμου (ΓΑ) για το σύνολο δεδομένων Balance-Scale . Παρατηρούμε ότι ο ΓΑ παρουσιάζει μία φθίνουσα πορεία όσον αφορά την τιμή του MSE αναλογικά με την αύξηση του αριθμού των επαναλήψεων . Η μικρότερη τιμή του MSE είναι περίπου ίση με 1.12 .



Εικόνα 45: πειραματικά αποτελέσματα του Γενετικού Αλγορίθμου (ΓΑ) για το σύνολο δεδομένων Car. Παρατηρούμε ότι ο ΓΑ δεν συγκλίνει σε μία συγκεκριμένη τιμή αναλογικά με τον αριθμό των επαναλήψεων . Αντίθετα, το MSE μεταβάλλεται ανάμεσα στις τιμές 0.70 και 0.74 . Αυτό πρακτικά σημαίνει ότι ο αλγόριθμος παγιδεύεται σε ένα τοπικό ελάχιστο του χώρου αναζήτησης .



Εικόνα 45: πειραματικά αποτελέσματα του Γενετικού Αλγορίθμου (ΓΑ) για το σύνολο δεδομένων Cloud . Παρατηρούμε ότι το MSE παρουσιάζει μία καθοδική γραμμική πορεία σε αναλογία με τον αριθμό των επαναλήψεων . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.00992 .



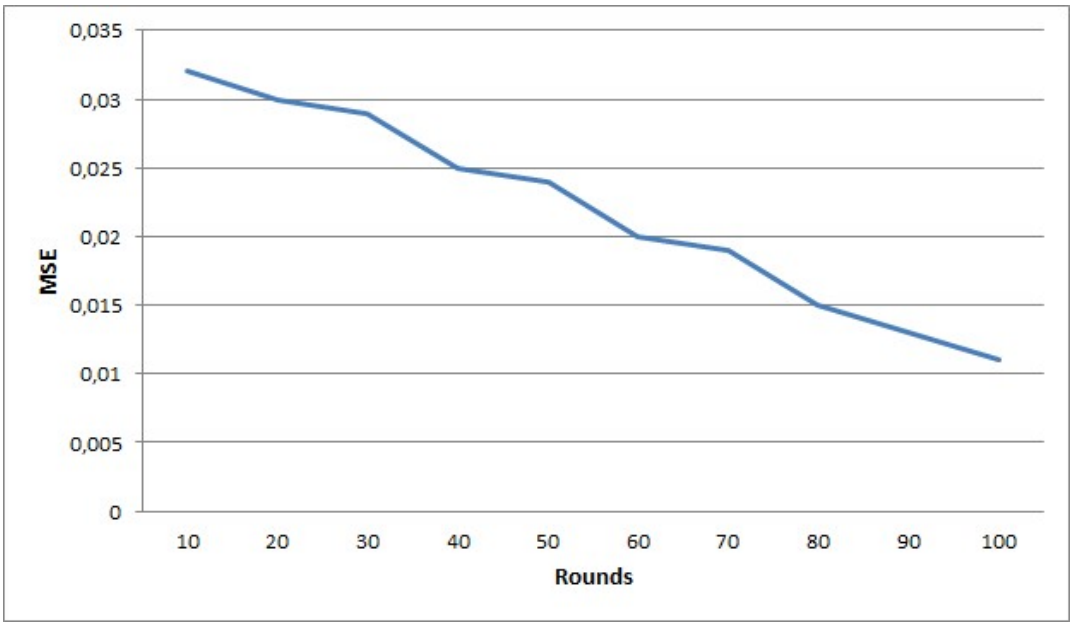
Εικόνα 46: πειραματικά αποτελέσματα του Γενετικού Αλγορίθμου (ΓΑ) για το σύνολο δεδομένων Iris . Παρατηρούμε ότι ο ΓΑ δεν συγκλίνει σε μία συγκεκριμένη τιμή αναλογικά με τον αριθμό των επαναλήψεων . Αντίθετα , το MSE μεταβάλλεται ανάμεσα στις τιμές 0.21 και 0.23 . Αυτό πρακτικά σημαίνει ότι ο αλγόριθμος παγιδεύεται σε ένα τοπικό ελάχιστο του χώρου αναζήτησης .

Σχολιασμός των πειραματικών αποτελεσμάτων

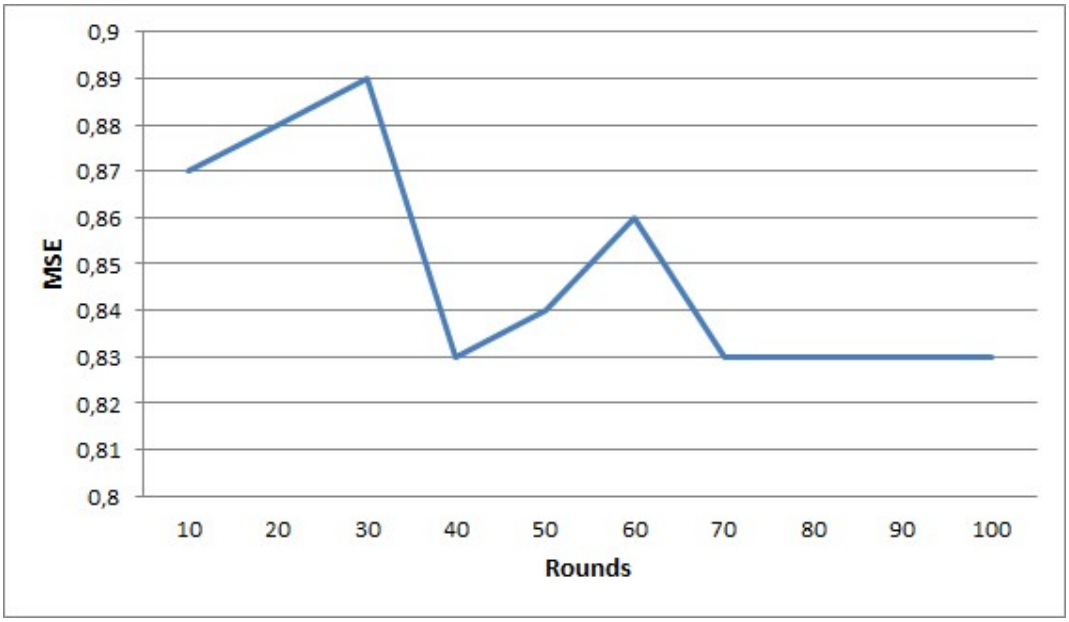
Ο Γενετικός Αλγόριθμος εμφανίζει αρκετά καλά αποτελέσματα στα σύνολα δεδομένων Abalone , Balance-Scale και Cloud ενώ δεν τα πηγαίνει και τόσο καλά στα σύνολα δεδομένων Car και Iris . Στα σύνολα δεδομένων Car και Iris είναι εμφανές ότι ο ΓΑ παγιδεύεται σε κάποιο τοπικό ελάχιστο , με αποτέλεσμα να μην είναι σε θέση να μειώσει το MSE ακόμα περισσότερο .

8.4 Πειραματικά αποτελέσματα του αλγορίθμου PSO

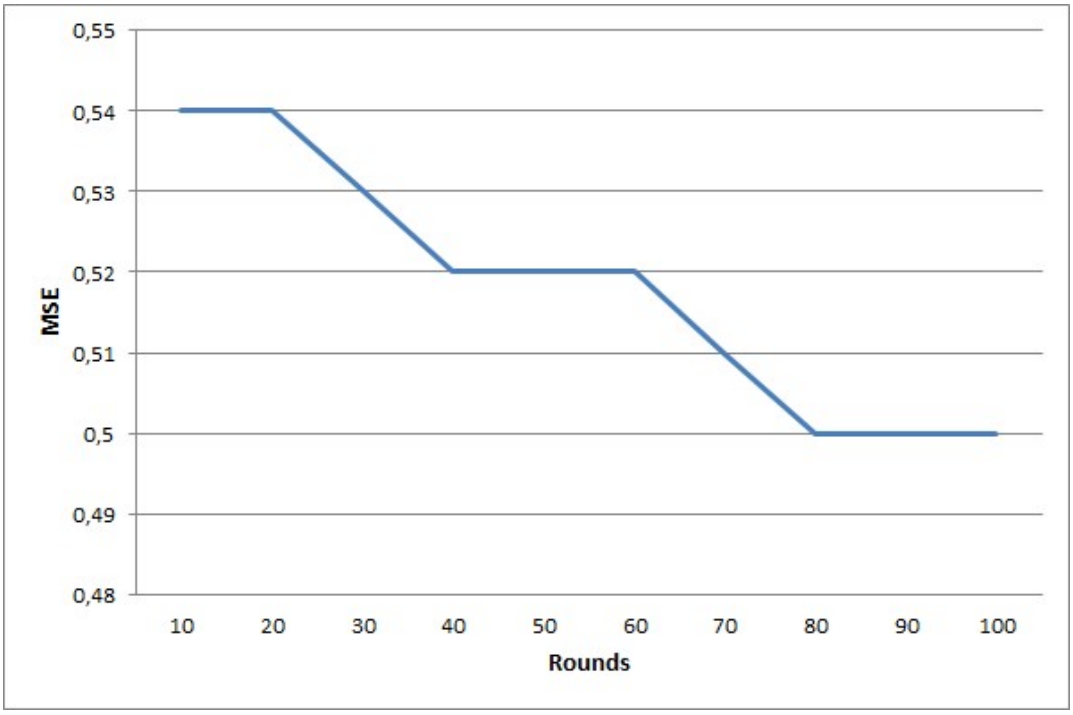
Τα πειραματικά αποτελέσματα για τον αλγόριθμο PSO παρουσιάζονται στις Εικόνες 48 – 52 .



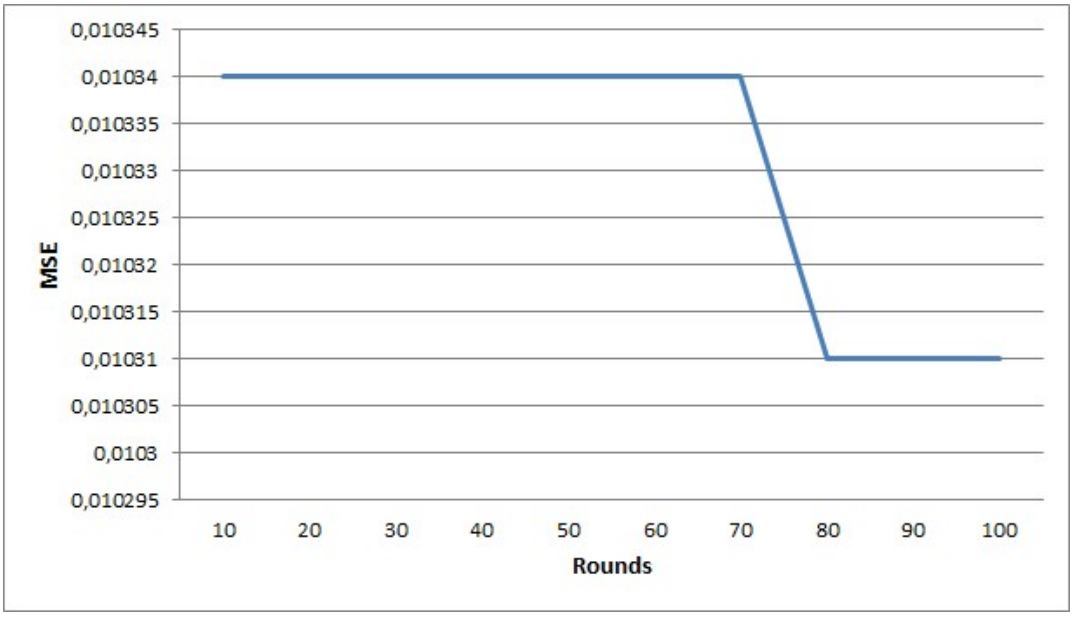
Εικόνα 48: πειραματικά αποτελέσματα του αλγορίθμου PSO για το σύνολο δεδομένων Abalone . Παρατηρούμε ότι ο PSO παρουσιάζει μία φθίνουσα πορεία όσον αφορά την τιμή του MSE αναλογικά με την αύξηση του αριθμού των επαναλήψεων . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.01 .



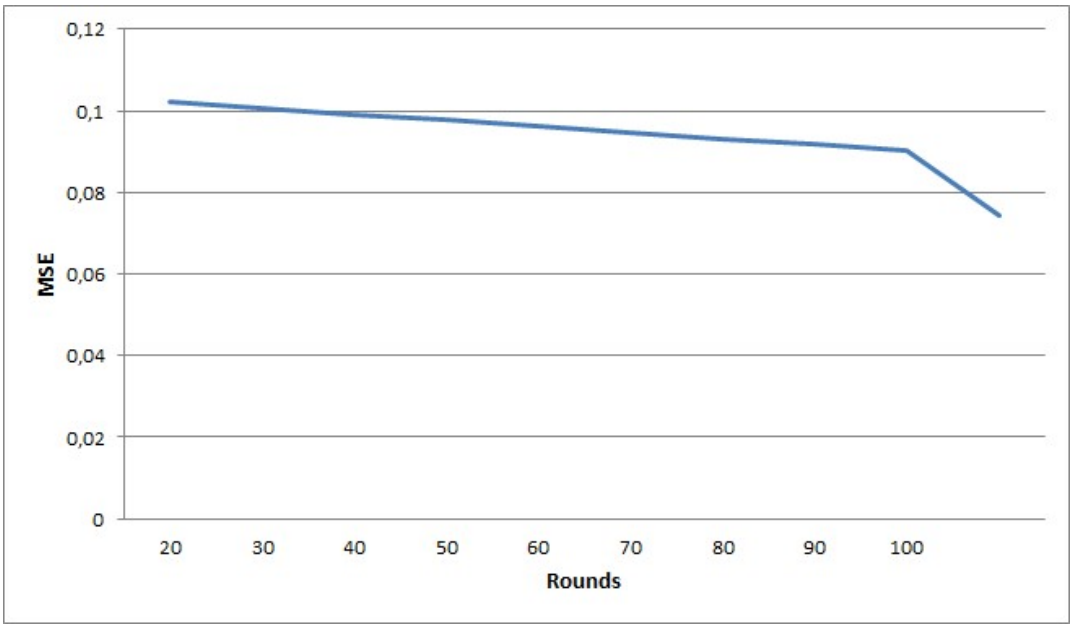
Εικόνα 49: πειραματικά αποτελέσματα του αλγορίθμου PSO για το σύνολο δεδομένων Balance-Scale . Παρατηρούμε ότι ο PSO συγκλίνει σε MSE ίσο με 0.83 όσο αυξάνεται ο αριθμός των επαναλήψεων . Επειδή η τιμή αυτή δεν είναι η βέλτιστη, μπορούμε να συμπεράνουμε ότι ο PSO παγιδεύτηκε σε κάποιο τοπικό ελάχιστο του χώρου αναζήτησης .



Εικόνα 50: πειραματικά αποτελέσματα του αλγορίθμου PSO για το σύνολο δεδομένων Car . Παρατηρούμε ότι ο PSO συγκλίνει σε MSE ίσο με 0.5 όσο αυξάνεται ο αριθμός των επαναλήψεων . Επειδή η τιμή αυτή δεν είναι η βέλτιστη , μπορούμε να συμπεράνουμε ότι ο PSO παγιδεύτηκε σε κάποιο τοπικό ελάχιστο του χώρου αναζήτησης .



Εικόνα 51: πειραματικά αποτελέσματα του αλγορίθμου PSO για το σύνολο δεδομένων Cloud . Παρατηρούμε ότι ο PSO συγκλίνει σε MSE ίσο με 0.01031 όσο αυξάνεται ο αριθμός των επαναλήψεων . Επειδή η τιμή αυτή δεν είναι η βέλτιστη , μπορούμε να συμπεράνουμε ότι ο PSO παγιδεύτηκε σε κάποιο τοπικό ελάχιστο του χώρου αναζήτησης .



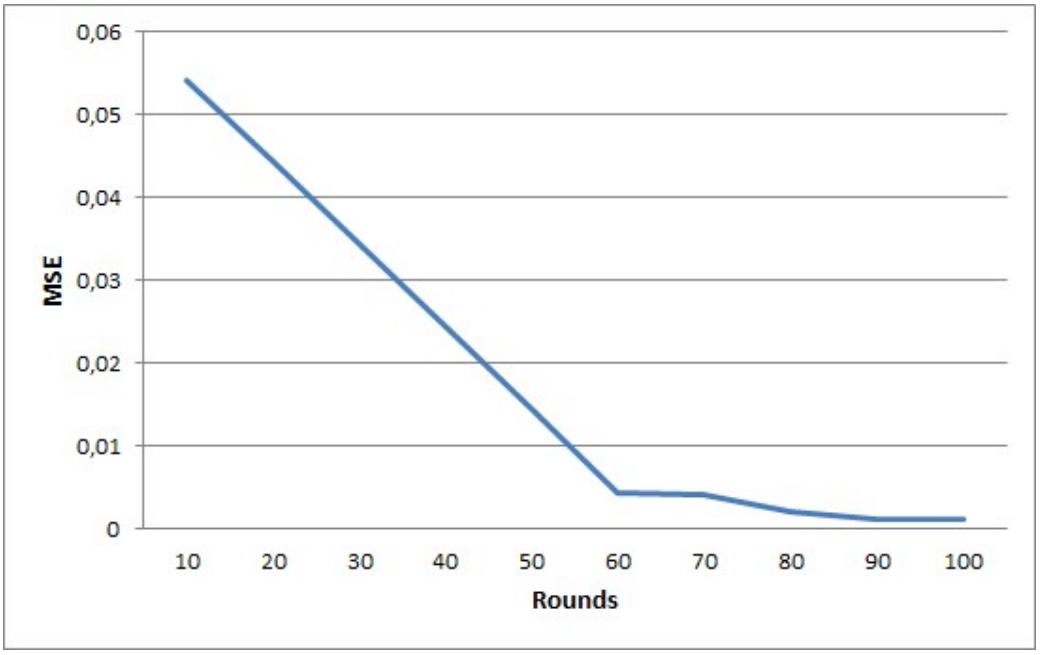
Εικόνα 52: πειραματικά αποτελέσματα του αλγορίθμου PSO για το σύνολο δεδομένων Iris . Παρατηρούμε ότι ο PSO παρουσιάζει μία φθίνουσα πορεία όσον αφορά την τιμή του MSE αναλογικά με την αύξηση του αριθμού των επαναλήψεων . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.07 .

Σχολιασμός των πειραματικών αποτελεσμάτων

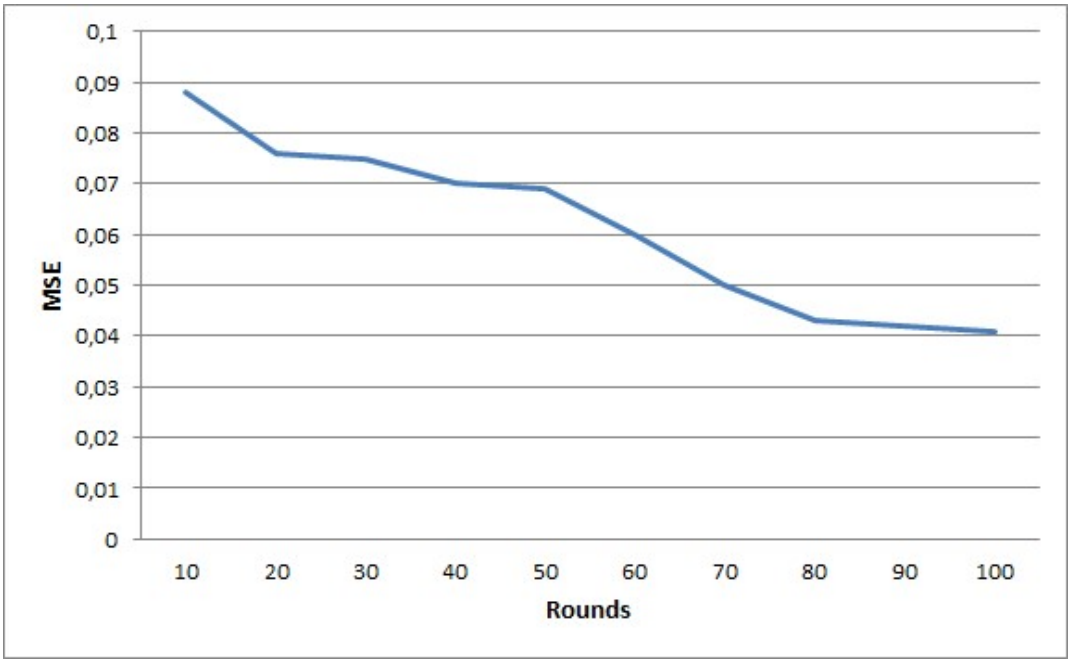
Ο αλγόριθμος PSO εμφανίζει αρκετά καλά αποτελέσματα στα σύνολα δεδομένων Abalone , Cloud και Iris ενώ δεν τα πηγαίνει και τόσο καλά στα σύνολα δεδομένων Balance-Scale και Car . Στα περισσότερα σύνολα δεδομένων είναι εμφανές ότι είναι αρκετά σύνηθες ο αλγόριθμος PSO να παγιδεύεται σε κάποιο τοπικό ελάχιστο του χώρου αναζήτησης . Αυτό μπορεί να εξηγηθεί από το γεγονός του ότι δεν διαθέτει κάποιον μηχανισμό παρόμοιο της μετάλλαξης του ΓΑ , έτσι ώστε να μπορέσει να ξεφύγει από τα τοπικά ελάχιστα .

8.5 Πειραματικά αποτελέσματα του αλγορίθμου HGAPSO

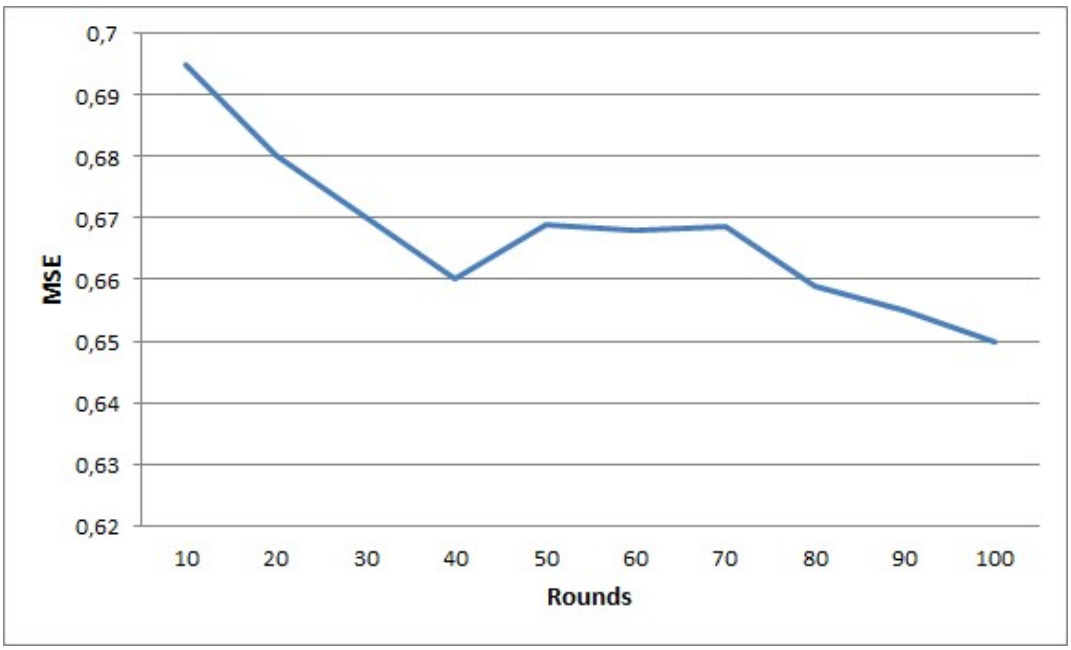
Τα πειραματικά αποτελέσματα για τον αλγόριθμο HGAPSO παρουσιάζονται στις Εικόνες 53 – 57 .



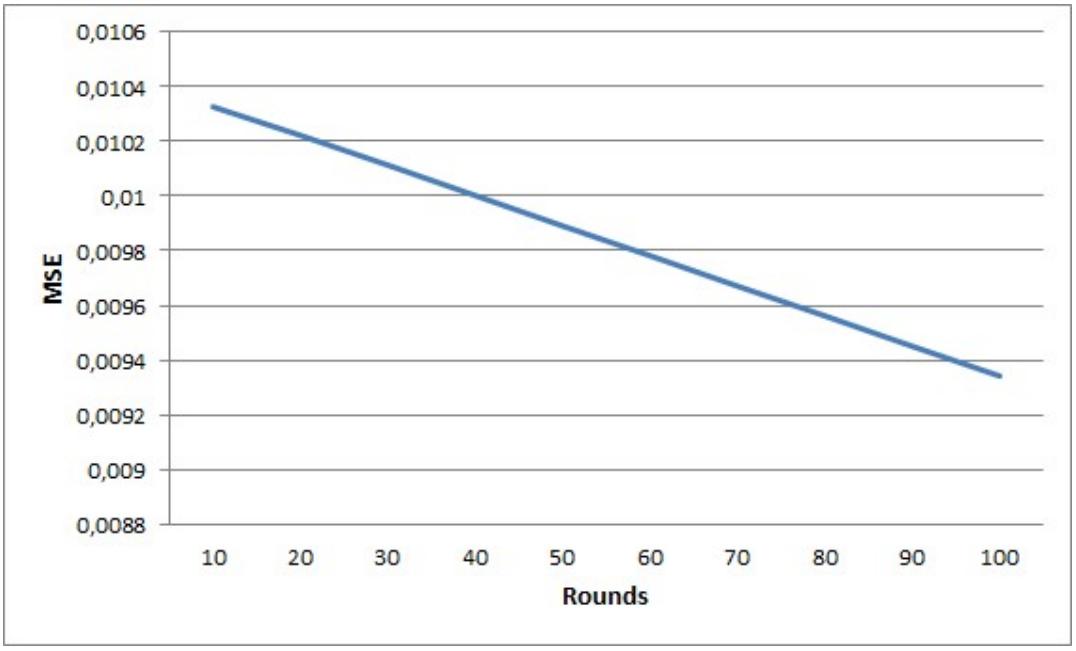
Εικόνα 53: πειραματικά αποτελέσματα του αλγορίθμου HGAPSO για το σύνολο δεδομένων Abalone . Παρατηρούμε ότι ο HGAPSO παρουσιάζει μία φθίνουσα πορεία όσον αφορά την τιμή του MSE αναλογικά με την αύξηση του αριθμού των επαναλήψεων . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0 .



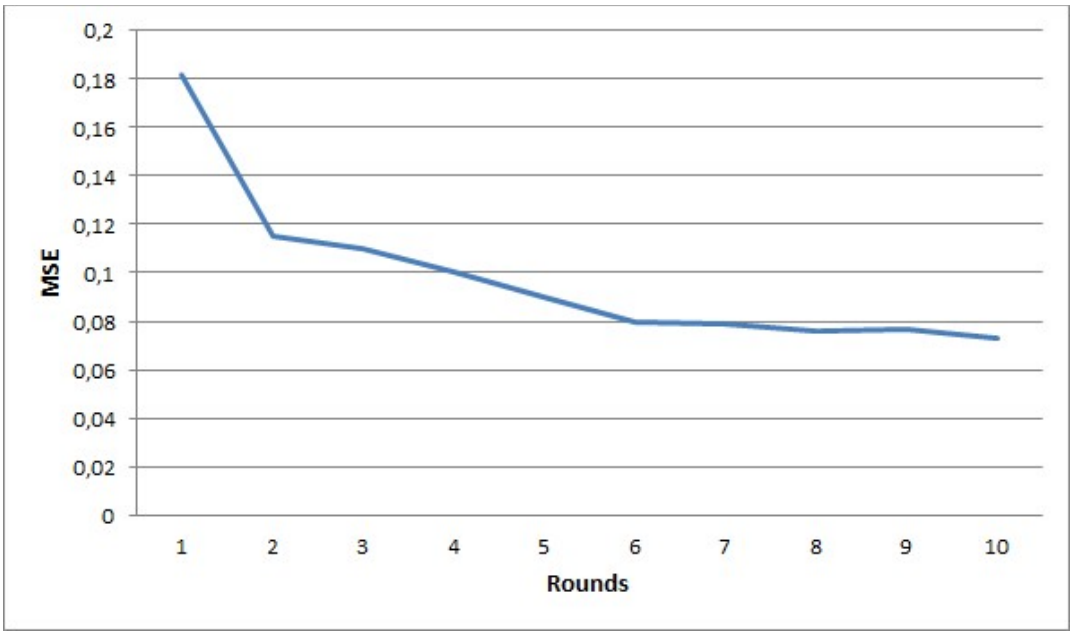
Εικόνα 54: πειραματικά αποτελέσματα του αλγορίθμου HGAPSO για το σύνολο δεδομένων Balance-Scale . Παρατηρούμε ότι ο HGAPSO παρουσιάζει μία φθίνουσα πορεία όσον αφορά την τιμή του MSE αναλογικά με την αύξηση του αριθμού των επαναλήψεων . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.05 .



Εικόνα 55: πειραματικά αποτελέσματα του αλγορίθμου HGAPSO για το σύνολο δεδομένων Car . Παρατηρούμε ότι ο HGAPSO συγκλίνει σε MSE ίσο με 0.65 όσο αυξάνεται ο αριθμός των επαναλήψεων . Επειδή η τιμή αυτή δεν είναι η βέλτιστη , μπορούμε να συμπεράνουμε ότι ο HGAPSO παγιδεύτηκε σε κάποιο τοπικό ελάχιστο του χώρου αναζήτησης .



Εικόνα 56: πειραματικά αποτελέσματα του αλγορίθμου HGAPSO για το σύνολο δεδομένων Cloud . Παρατηρούμε ότι ο HGAPSO παρουσιάζει μία φθίνουσα πορεία όσον αφορά την τιμή του MSE αναλογικά με την αύξηση του αριθμού των επαναλήψεων . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.091 .



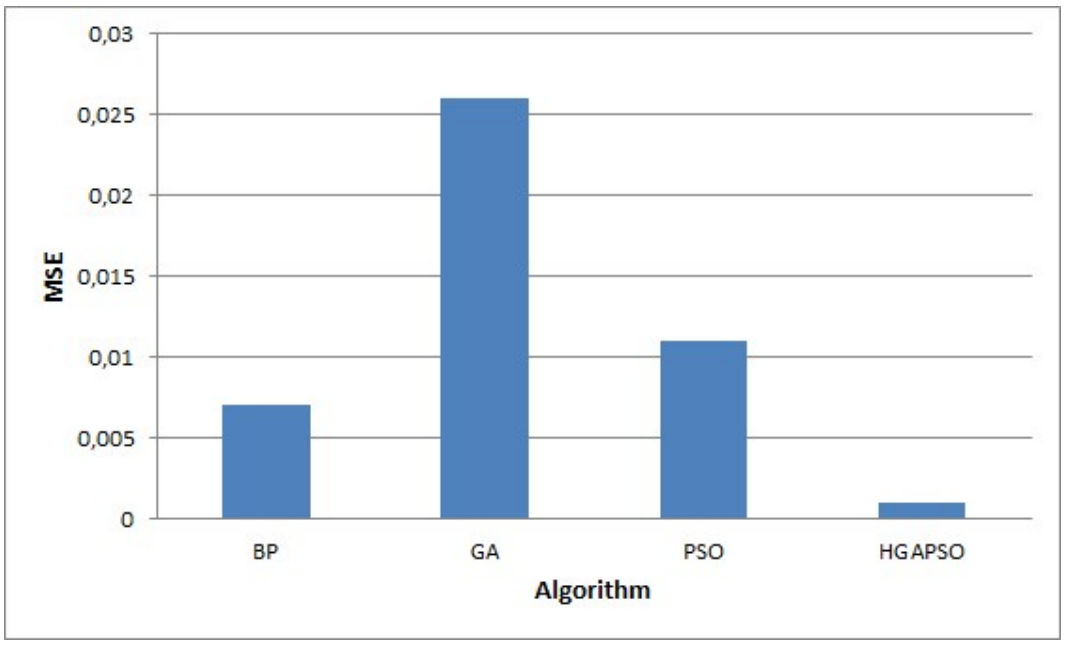
Εικόνα 57: πειραματικά αποτελέσματα του αλγορίθμου HGAPSO για το σύνολο δεδομένων Iris . Παρατηρούμε ότι ο HGAPSO παρουσιάζει μία φθίνουσα πορεία όσον αφορά την τιμή του MSE αναλογικά με την αύξηση του αριθμού των επαναλήψεων . Η μικρότερη τιμή του MSE είναι περίπου ίση με 0.07 .

Σχολιασμός των πειραματικών αποτελεσμάτων

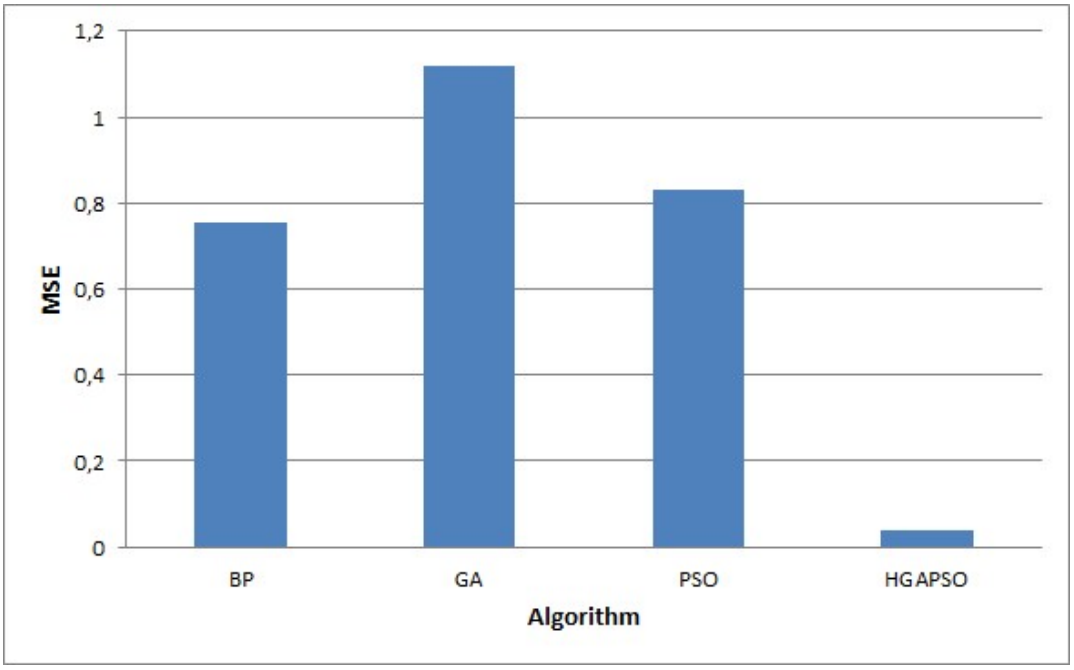
Ο αλγόριθμος HGAPSO εμφανίζει πολύ καλά αποτελέσματα σε όλα τα σύνολα δεδομένων με μόνη εξαίρεση το σύνολο δεδομένων Car . Στα περισσότερα σύνολα δεδομένων είναι εμφανές ότι ο αλγόριθμος HGAPSO καταφέρνει να μην παγιδευτεί σε κάποιο τοπικό βέλτιστο του χώρου αναζήτησης . Αυτό μπορεί να εξηγηθεί από το γεγονός του ότι χρησιμοποιεί τόσο τις ιδιότητες του Γενετικού Αλγορίθμου (μετάλλαξη) όσο και τις ιδιότητες του αλγορίθμου PSO .

8.6 Σύγκριση των 4 αλγορίθμων

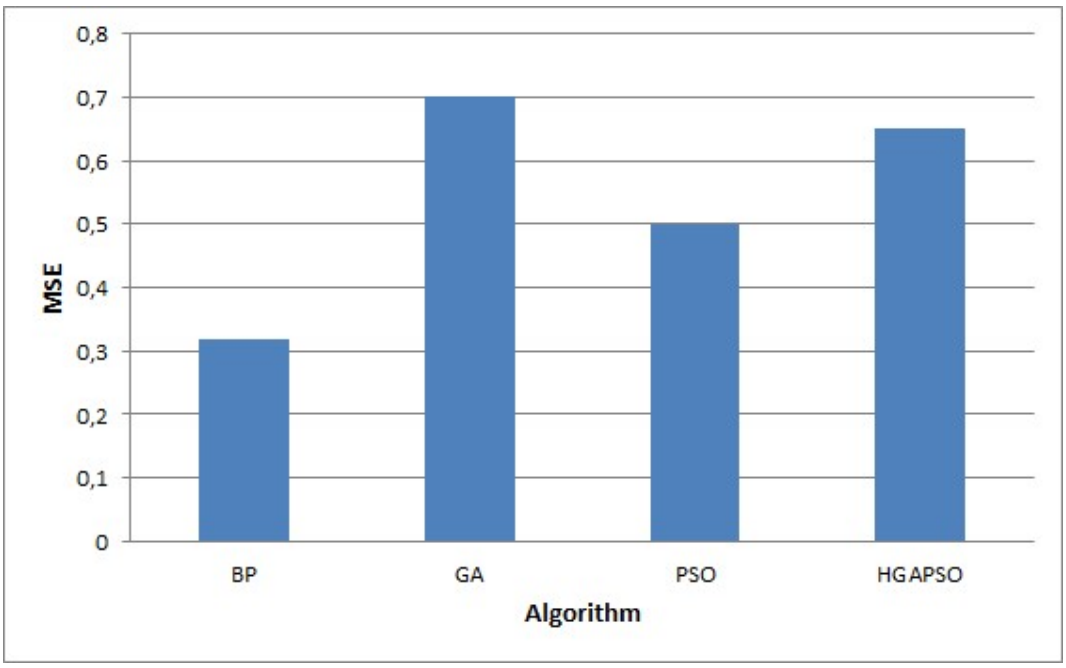
Τα αποτελέσματα για κάθε σύνολο δεδομένων παρουσιάζονται συγκεντρωτικά παρακάτω και για τους 4 αλγορίθμους . Ειδικότερα , για κάθε σύνολο δεδομένων επιλέγεται η καλύτερη επίδοση του κάθε αλγορίθμου , με βάση τα πειράματα των αμέσως προηγούμενων ενοτήτων . Τα αποτελέσματα παρουσιάζονται στις Εικόνες 58 – 62 .



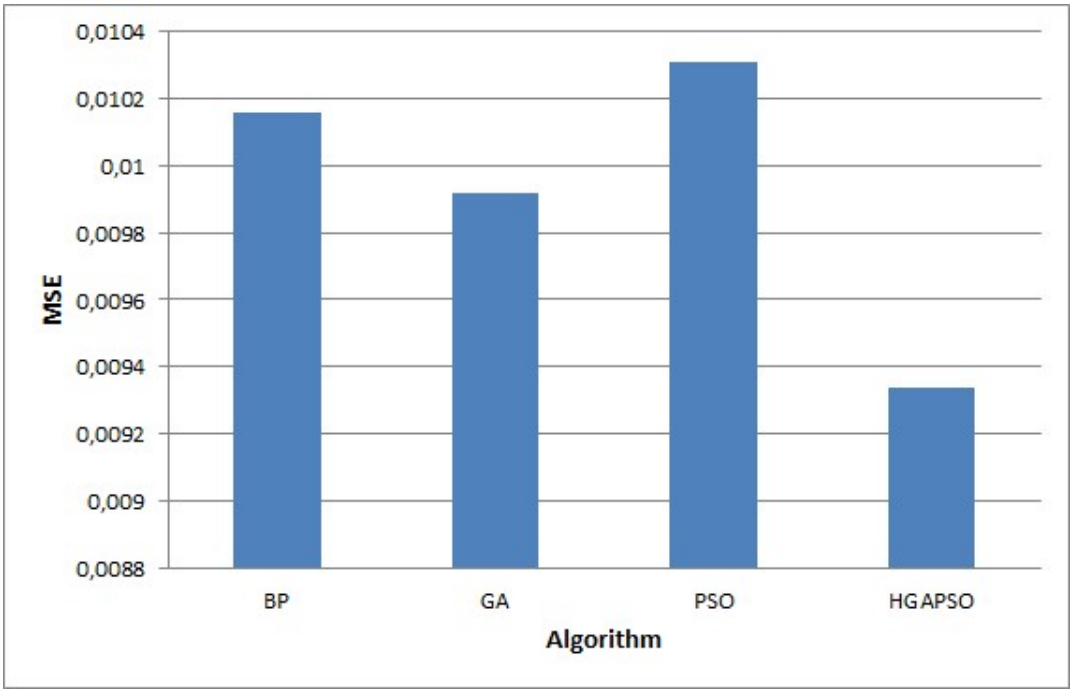
Εικόνα 58: σύγκριση των 4 αλγορίθμων για το σύνολο δεδομένων Abalone . Ο αλγόριθμος HGAPSO παρουσιάζει την καλύτερη επίδοση , με δεύτερο τον αλγόριθμο BP . Στην συνέχεια ακολουθεί ο αλγόριθμος PSO με την χειρότερη επίδοση να την έχει ο ΓΑ .



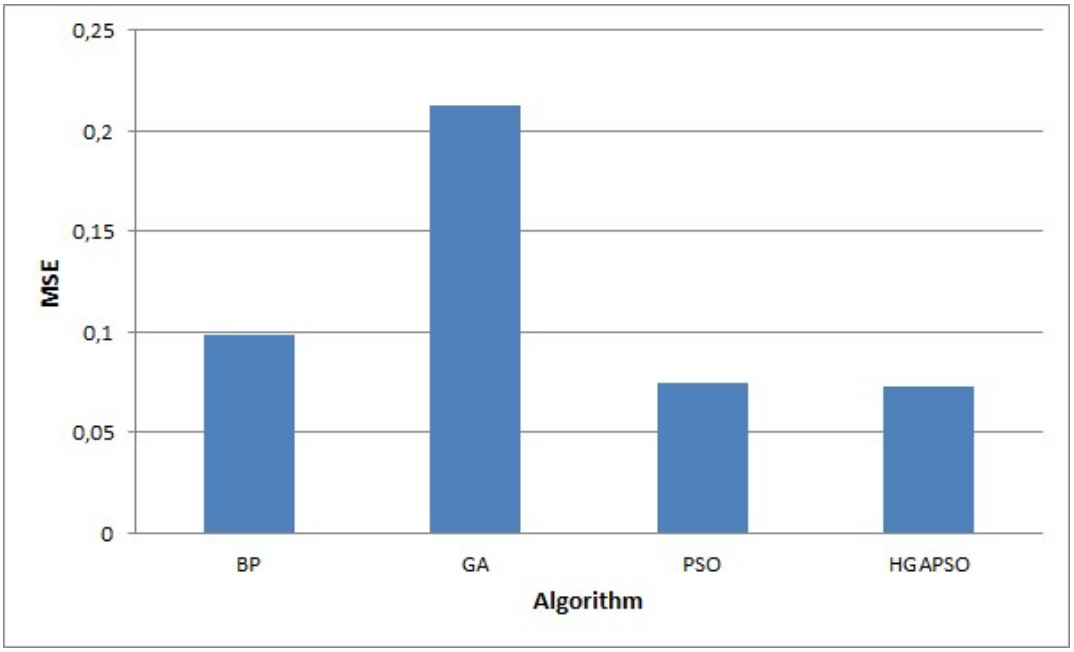
Εικόνα 59: σύγκριση των 4 αλγορίθμων για το σύνολο δεδομένων Balance-Scale . Ο αλγόριθμος HGAPSO παρουσιάζει την καλύτερη επίδοση , με δεύτερο τον αλγόριθμο BP . Στην συνέχεια ακολουθεί ο αλγόριθμος PSO με την χειρότερη επίδοση να την έχει ο ΓΑ .



Εικόνα 60: σύγκριση των 4 αλγορίθμων για το σύνολο δεδομένων Car . Ο αλγόριθμος BP παρουσιάζει την καλύτερη επίδοση , με δεύτερο τον αλγόριθμο PSO . Στην συνέχεια ακολουθεί ο αλγόριθμος HGAPSO με την χειρότερη επίδοση να την έχει ο ΓΑ .



Εικόνα 61: σύγκριση των 4 αλγορίθμων για το σύνολο δεδομένων Cloud . Ο αλγόριθμος HGAPSO παρουσιάζει την καλύτερη επίδοση , με δεύτερο τον ΓΑ . Στην συνέχεια ακολουθεί ο αλγόριθμος BP με την χειρότερη επίδοση να την έχει ο αλγόριθμος PSO .



Εικόνα 62: σύγκριση των 4 αλγορίθμων για το σύνολο δεδομένων Iris . Ο αλγόριθμος HGAPSO παρουσιάζει την καλύτερη επίδοση , με δεύτερο τον αλγόριθμο PSO . Στην συνέχεια ακολουθεί ο αλγόριθμος BP με την χειρότερη επίδοση να την έχει ο ΓΑ .

Σχολιασμός των πειραματικών αποτελεσμάτων

Από τα παραπάνω συγκριτικά διαγράμματα είναι εμφανές ότι ο HGAPSO παρουσιάζει τα καλύτερα πειραματικά αποτελέσματα , με τους αλγορίθμους BP και PSO να ακολουθούν κατά πόδας . Ο αλγόριθμος BP εμφανίζει ελαφρώς καλύτερα αποτελέσματα από τον αλγόριθμο PSO στην μέση περίπτωση . Τέλος , ο ΓΑ εμφανίζεται να έχει τις χειρότερες επιδόσεις , σε όλα σχεδόν τα σύνολα δεδομένων .

ΚΕΦΑΛΑΙΟ 9: ΣΥΜΠΕΡΑΣΜΑΤΑ – ΕΠΕΚΤΑΣΕΙΣ

9.1 Συμπεράσματα

Στην παρούσα διπλωματική εργασία διεκπεραιώσαμε τους παρακάτω στόχους :

1. Μελετήσαμε τους εξελικτικούς αλγορίθμους και περισσότερο συγκεκριμένα τον Γενετικό Αλγόριθμο (ΓΑ) και τον αλγόριθμο PSO .
2. Μελετήσαμε τα Τεχνητά Νευρωνικά Δίκτυα (ΤΝΔ) και είδαμε τον συμβατικό τρόπο εκπαίδευσής τους (αλγόριθμος Οπισθοδιάδοσης του Σφάλματος – BP) .
3. Μελετήσαμε τον τρόπο εκπαίδευσης ενός ΤΝΔ με την χρήση εξελικτικών αλγορίθμων .
4. Ορίσαμε τον υβριδικό εξελικτικό αλγόριθμο HGAPSO , ο οποίος δανείζεται στοιχεία τόσο από τον ΓΑ όσο και από τον αλγόριθμο PSO .
5. Υλοποιήσαμε λογισμικό στην γλώσσα προγραμματισμού Java με το οποίο μπορούμε να εκτελέσουμε και τους 4 αλγορίθμους (BP , ΓΑ , PSO , HGAPSO) πάνω σε δεδομένα εκπαίδευσης .
6. Εφαρμόσαμε και τις 4 τεχνικές (BP , ΓΑ , PSO , HGAPSO) σε 5 σύνολα δεδομένων με δεδομένα κατηγοριοποίησης και καταγράψαμε τα πειραματικά αποτελέσματα . Ως κριτήριο σύγκρισης χρησιμοποιήσαμε το Μέσο Τετραγωνικό Σφάλμα (MSE) .

Από την πειραματική μελέτη των αποτελεσμάτων καταλήξαμε στα εξής συμπεράσματα :

1. Ο αλγόριθμος HGAPSO είναι σε θέση να εκπαιδεύσει ένα ΤΝΔ συγκεκριμένης αρχιτεκτονικής με μικρό ποσοστό λάθους (MSE) .
2. Ο ΓΑ αποδεικνύεται ως χειρότερη επιλογή όσον αφορά και τους 4 αλγορίθμους .
3. Δεν υπάρχει καλύτερη καθολική λύση όσον αφορά την επιλογή του αλγορίθμου εκπαίδευσης , αφού αυτό εξαρτάται αρκετά από τα δεδομένα εκπαίδευσης .

9.2 Επεκτάσεις

Η παρούσα διπλωματική εργασία θα μπορούσε να επεκταθεί με τουλάχιστον έναν από τους παρακάτω τρόπους :

- Υλοποίηση περισσότερων εξελικτικών αλγορίθμων στο λογισμικό .
- Οπτικοποίηση των αποτελεσμάτων στο λογισμικό .
- Χρήση περισσότερων τελεστών διασταύρωσης και μετάλλαξης στον Γενετικό Αλγόριθμο .
- Τήρηση αρχείου με το ιστορικό των συνόλων δεδομένων που εξετάσθηκαν , έτσι ώστε να είναι δυνατή η παραγωγή στατιστικών δεδομένων .
- Εξέταση περισσότερων συνόλων δεδομένων .
- Σύγκριση των αλγορίθμων με περισσότερα κριτήρια από το MSE .

ΒΙΒΛΙΟΓΡΑΦΙΚΕΣ ΑΝΑΦΟΡΕΣ

Abido A. (2001). “Particle swarm optimization for multimachine power system stabilizer design”, in Proc. Power Engineering Society Summer Meeting (PES), vol. 3, pp. 1346-1351.

Al-kazemi B., Mohan C. (2002). "Training feedforward neural networks using multi-phase particle swarm optimization", Proceedings of the 9th International Conference on Neural Information Processing, pp.2615-2619.

Cybenko G. (1989). "Approximations by superpositions of sigmoidal functions", Mathematics of Control, Signals, and Systems, vol. 2, pp. 303-314.

Eberhart R., Shi Y. (1998). "Comparison between genetic algorithms and particle swarm optimization", Proceedings of the 7th International Conference on Evolutionary Programming, vol. 7, pp.611-616.

Eberhart R., Shi Y., Kennedy J. (2001). “Swarm Intelligence”, San Mateo, CA: Morgan Kaufmann. Goldberg D. E. (1989). “Genetic Algorithms in Search, Optimization and Machine Learning”, Addison Wesley.Holland J. (1992). “Genetic Algorithms”, Scientific American, pp. 44-50.Juang C. F. (2004). "A hybrid of genetic algorithm and particle swarm optimization for recurrent network design", Systems, Man, and Cybernetings, vol. 34, no. 2, pp.997-1006.

Kennedy J., Eberhart R.C. (1995). “Particle swarm optimization”, Proceedings of the IEEE International JointConference on Neural Networks, IEEE Press, pp. 1942-1948. Koza J., Rice J. (1991). "Genetic Generation of Both the Weight and Architecture for a Neural Network", Proceedings of the International Joint Conference on Neural Networks, vol. 2, pp.397-404.

Mendes R., Cortez P., Rocha M., Neves J. (2002). "Particle swarms for feedforward neural network training", Proceedings of the 2002 International Joing Conference on Neural Networks, pp.1895-1899.

Montana D., Davis L. (1989). "Training feedforward neural networks using genetic algorithms", Proceedings of the 11th International Joint Conference on Artificial Intelligence, pp. 762-767.

Ozcan E., Mohan C. (1999). “Particle swarm optimization: Surfing the waves”, in Proc. IEEE Congress Evol. Comput., vol. 3, pp. 1939-1944.

Richards M., Ventura D. (2004). “Choosing a starting configuration for particle swarm optimization”, in Proc. IEEE Int. Joint. Conf. Neural Ne., vol. 3, pp. 2309-2312.

Rojas R. (1996). "Neural Networks: A Systematic Introduction". Springer-Vertag Press, Berlin.

Rosenblatt F. (1958).“The perceptron: A probabilistic model for information storage and organization in the brain”, Psychological Review, Vol. 65, pp. 386–408.

Schiffmann W., Joost M., Werner R. (1991). "Performance Evaluation of Evolutionary Created Neural Network Topologies", Parallel Problem Solving from Natur 2, pp. 292-296.

Schiffmann W., Joost M., Werner R. (1992). "Synthesis and Performance Analysis of Neural Network Architectures", Techical Report 16/1992, University of Koblenz, Germany.

Schiffmann W., Joost M., Werner R. (1993). "Optimization of the Backpropagation Algorithm for Training Multilayer Perceptrors", Proceedings of the International Joint Conference on Neural Networks and Genetic Algorithms, pp. 675-682.

Streichert F. (2002). “Introduction to Evolutionary Algorithms”, In proceedings of the Frankfurt MathFinance Workshop, Frankfurt, Germany, pp. 1-8.

Valle Y., Venayagamoorthy, G.K., Mohagheghi S., Hernandez J.C.(2008). “Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems”, IEEE Transactions on Evolutionary Computation, vol. 12, No. 2, pp. 171-195.

Van den Bergh F., Engelbrecht A. (2004). "A Cooperative approach to particle swarm optimization", Evolutionary Computation, vol. 8, pp. 225-239.

White D. (1993). "GANNet: A genetic Algorithm for Searching Topology and Weight Spaces in Neural Network Design", Springer, Lecture Notes in Computer Science, vol. 686, 1993, pp. 322-327

Whitley D., Starkweather T., Bogart C. (1990). "Genetic algorithms and neural networks: optimizing connections and connectivity", Parallel Computing 14, pp. 347-361.

Xiaorong P., Leiting C., Zhongjie F. (2007). "PSO-Based Adaptive Person Identification With Face and Fingerprint Fusion", Advances in Neural Networks, vol. 14, pp. 481-4855.

ΠΑΡΑΡΤΗΜΑ Α: ΚΩΔΙΚΑΣ ΤΟΥ ΛΟΓΙΣΜΙΚΟΥ

```
package forms;

import evaluation.ResultEvaluator;
import hgapso.GeneticAlgorithm;
import hgapso.HGAPSOAlgorithm;
import hgapso.PSOAlgorithm;
import java.io.File;
import java.text.DecimalFormat;
import java.util.ArrayList;
import javax.swing.DefaultComboBoxModel;
import javax.swing.SwingWorker;
import neuralnetwork.NeuralNetwork;
import neuralnetwork.TrainingSet;
import trainingalgorithms.BackPropagationAlgorithm;

public class TrainingSetSelector extends javax.swing.JFrame {

   private ArrayList<File> trainingSetFiles;
   private DecimalFormat df;

   public TrainingSetSelector() {
      initComponents();
      this.readTrainingFiles();
      df = new DecimalFormat("#.#####");
   }

   private void readTrainingFiles() {

      File folder = new File("trainingsets");
      DefaultComboBoxModel cbModel = new DefaultComboBoxModel();
      trainingSetComboBox.setModel(cbModel);
      cbModel.removeAllElements();
      trainingSetFiles = new ArrayList<>();
      for (File f : folder.listFiles()) {
         if (!f.isDirectory()) {
            cbModel.addElement(f.getName());
            trainingSetFiles.add(f);
         }
      }

   }
```

```
    /**
     * This method is called from within the constructor to initialize the form.
     * WARNING: Do NOT modify this code. The content of this method is always
     * regenerated by the Form Editor.
     */
    @SuppressWarnings("unchecked")
    // <editor-fold defaultstate="collapsed" desc="Generated Code">
    private void initComponents() {

        trainingSetLabel = new javax.swing.JLabel();
        trainingSetComboBox = new javax.swing.JComboBox();
        algorithmLabel = new javax.swing.JLabel();
        algorithmComboBox = new javax.swing.JComboBox();
        hgapsoPanel = new javax.swing.JPanel();
        hgapsoRoundsLabel = new javax.swing.JLabel();
        hgapsoRoundsSpinner = new javax.swing.JSpinner();
        hgapsoPopSizeLabel = new javax.swing.JLabel();
        hgapsoPopSizeSpinner = new javax.swing.JSpinner();
        gaPanel = new javax.swing.JPanel();
        pcLabel = new javax.swing.JLabel();
        pmLabel = new javax.swing.JLabel();
        pcSpinner = new javax.swing.JSpinner();
        pmSpinner = new javax.swing.JSpinner();
        gaRoundsLabel = new javax.swing.JLabel();
        gaRoundsSpinner = new javax.swing.JSpinner();
        gaPopSizeLabel = new javax.swing.JLabel();
        gaPopSizeSpinner = new javax.swing.JSpinner();
        bpPanel = new javax.swing.JPanel();
        bpEpochsLabel = new javax.swing.JLabel();
        bpEpochsSpinner = new javax.swing.JSpinner();
        resultsAreaScrollPane = new javax.swing.JScrollPane();
        output = new javax.swing.JTextArea();
        neuralNetworkPanel = new javax.swing.JPanel();
        layersNoLabel = new javax.swing.JLabel();
        nodesPerLayerLabel = new javax.swing.JLabel();
        layersNoSpinner = new javax.swing.JSpinner();
        nodesPerLayerSpinner = new javax.swing.JSpinner();
        resultsLabel = new javax.swing.JLabel();
        executeButton = new javax.swing.JButton();
        psoPanel = new javax.swing.JPanel();
        c1Label = new javax.swing.JLabel();
        c2Label = new javax.swing.JLabel();
        c1Spinner = new javax.swing.JSpinner();
        c2Spinner = new javax.swing.JSpinner();
```

```
psoRoundsLabel = new javax.swing.JLabel();
psoRoundsSpinner = new javax.swing.JSpinner();
psoPopSizeLabel = new javax.swing.JLabel();
psoPopSizeSpinner = new javax.swing.JSpinner();

setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
setTitle("Neural Network Training Expirements");

trainingSetLabel.setText("Training Set:");

trainingSetComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Item 1", "Item 2", "Item 3", "Item 4" }));

algorithmLabel.setText("Algorithm:");

algorithmComboBox.setModel(new javax.swing.DefaultComboBoxModel(new String[] { "Back Propagation Algorithm", "Genetic Algorithm", "PSO (Particle Swarm Optimization)", "HGAPSO (Hybrid Genetic Algorithm Particle Swarm Optimization)" }));


hgapsoPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.createEtchedBorder(), "HGAPSO Algorithm Parameters"));

hgapsoRoundsLabel.setText("Rounds:");

hgapsoRoundsSpinner.setModel(new javax.swing.SpinnerNumberModel(10, 10, 10000, 10));

hgapsoPopSizeLabel.setText("Population size:");

hgapsoPopSizeSpinner.setModel(new javax.swing.SpinnerNumberModel(50, 10, 1000, 10));

javax.swing.GroupLayout hgapsoPanelLayout = new javax.swing.GroupLayout(hgapsoPanel);
hgapsoPanel.setLayout(hgapsoPanelLayout);
hgapsoPanelLayout.setHorizontalGroup(
  hgapsoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
  .addGroup(hgapsoPanelLayout.createSequentialGroup()
    .addContainerGap()
    .addComponent(hgapsoPopSizeLabel)
    .addGap(18, 18, 18)
    .addComponent(hgapsoPopSizeSpinner, javax.swing.GroupLayout.PREFERRED_SIZE, 64, javax.swing.GroupLayout.PREFERRED_SIZE)
    .addGap(43, 43, 43)
    .addComponent(hgapsoRoundsLabel)
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
            .addComponent(hgapsoRoundsSpinner, javax.swing.GroupLayout.PREFERRED_SIZE, 60,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
      );
      hgapsoPanelLayout.setVerticalGroup(
        hgapsoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(hgapsoPanelLayout.createSequentialGroup()
          .addContainerGap()

.addGroup(hgapsoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(hgapsoRoundsLabel)
            .addComponent(hgapsoRoundsSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(hgapsoPopSizeLabel)
            .addComponent(hgapsoPopSizeSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addContainerGap(17, Short.MAX_VALUE))
      );


gaPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.createEtch
edBorder(), "Genetic Algorithm Parameters"));

      pcLabel.setText("Crossover Probability:");

      pmLabel.setText("Mutation Probability:");

      pcSpinner.setModel(new javax.swing.SpinnerNumberModel(0.5d, 0.0d, 1.0d, 0.1d));

      pmSpinner.setModel(new javax.swing.SpinnerNumberModel(0.5d, 0.0d, 1.0d, 0.1d));

      gaRoundsLabel.setText("Rounds:");

      gaRoundsSpinner.setModel(new javax.swing.SpinnerNumberModel(10, 10, 1000, 10));

      gaPopSizeLabel.setText("Population size:");

      gaPopSizeSpinner.setModel(new javax.swing.SpinnerNumberModel(50, 10, 1000, 10));

      javax.swing.GroupLayout gaPanelLayout = new javax.swing.GroupLayout(gaPanel);
      gaPanel.setLayout(gaPanelLayout);
      gaPanelLayout.setHorizontalGroup(
        gaPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(gaPanelLayout.createSequentialGroup()
          .addContainerGap()
```

```
            .addComponent(pcLabel)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(pcSpinner, javax.swing.GroupLayout.PREFERRED_SIZE, 66,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
            .addComponent(pmLabel)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(pmSpinner, javax.swing.GroupLayout.PREFERRED_SIZE, 64,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(18, 18, 18)
            .addComponent(gaPopSizeLabel)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(gaPopSizeSpinner, javax.swing.GroupLayout.PREFERRED_SIZE, 54,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addGap(45, 45, 45)
            .addComponent(gaRoundsLabel)
            .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
            .addComponent(gaRoundsSpinner, javax.swing.GroupLayout.PREFERRED_SIZE, 63,
javax.swing.GroupLayout.PREFERRED_SIZE)
            .addContainerGap())
        );
        gaPanelLayout.setVerticalGroup(
          gaPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(gaPanelLayout.createSequentialGroup()
            .addContainerGap()
            .addGroup(gaPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
              .addComponent(pcLabel)
              .addComponent(pmLabel)
              .addComponent(pcSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addComponent(pmSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addComponent(gaRoundsLabel)
              .addComponent(gaRoundsSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
              .addComponent(gaPopSizeLabel)
              .addComponent(gaPopSizeSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addContainerGap(14, Short.MAX_VALUE))
        );


bpPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.createEtch
edBorder(), "Back Propagation Algorithm Parameters"));


        bpEpochsLabel.setText("Number of Epochs:");
```

```
bpEpochsSpinner.setModel(new javax.swing.SpinnerNumberModel(10, 10, 10000, 10));

javax.swing.GroupLayout bpPanelLayout = new javax.swing.GroupLayout(bpPanel);
bpPanel.setLayout(bpPanelLayout);
bpPanelLayout.setHorizontalGroup(
  bpPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
  .addGroup(bpPanelLayout.createSequentialGroup()
    .addContainerGap()
    .addComponent(bpEpochsLabel)
    .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
    .addComponent(bpEpochsSpinner, javax.swing.GroupLayout.PREFERRED_SIZE, 54,
javax.swing.GroupLayout.PREFERRED_SIZE)
    .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
);
bpPanelLayout.setVerticalGroup(
  bpPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
  .addGroup(javax.swing.GroupLayout.Alignment.TRAILING,
bpPanelLayout.createSequentialGroup()
    .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
    .addGroup(bpPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
      .addComponent(bpEpochsLabel)
      .addComponent(bpEpochsSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
    .addContainerGap())
);

output.setColumns(20);
output.setRows(5);
resultsAreaScrollPane.setViewportView(output);


neuralNetworkPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactor
y.createEtchedBorder(), "Neural Network Architecture"));

layersNoLabel.setText("Layers:");

nodesPerLayerLabel.setText("Noder per layer:");

layersNoSpinner.setModel(new javax.swing.SpinnerNumberModel(1, 1, 100, 1));

nodesPerLayerSpinner.setModel(new javax.swing.SpinnerNumberModel(5, 1, 100, 1));

javax.swing.GroupLayout neuralNetworkPanelLayout = new
javax.swing.GroupLayout(neuralNetworkPanel);
```

```
        neuralNetworkPanel.setLayout(neuralNetworkPanelLayout);
        neuralNetworkPanelLayout.setHorizontalGroup(
            neuralNetworkPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(neuralNetworkPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(layersNoLabel)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
                .addComponent(layersNoSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(nodesPerLayerLabel)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(nodesPerLayerSpinner, javax.swing.GroupLayout.PREFERRED_SIZE, 53,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap())
        );
        neuralNetworkPanelLayout.setVerticalGroup(
            neuralNetworkPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(neuralNetworkPanelLayout.createSequentialGroup()
                .addGap(16, 16, 16)

.addGroup(neuralNetworkPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELI
NE)
                    .addComponent(layersNoLabel)
                    .addComponent(nodesPerLayerLabel)
                    .addComponent(layersNoSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
                    .addComponent(nodesPerLayerSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
                .addContainerGap(22, Short.MAX_VALUE))
        );

        resultsLabel.setText("Results:");

        executeButton.setText("Execute");
        executeButton.addActionListener(new java.awt.event.ActionListener() {
            public void actionPerformed(java.awt.event.ActionEvent evt) {
                executeButtonActionPerformed(evt);
            }
        });


psoPanel.setBorder(javax.swing.BorderFactory.createTitledBorder(javax.swing.BorderFactory.createEtc
hedBorder(), "PSO Algorithm Parameters"));
```

```
        c1Label.setText("c1:");

        c2Label.setText("c2:");

        c1Spinner.setModel(new javax.swing.SpinnerNumberModel(0.5d, 0.0d, 1.0d, 0.1d));

        c2Spinner.setModel(new javax.swing.SpinnerNumberModel(0.5d, 0.0d, 1.0d, 0.1d));

        psoRoundsLabel.setText("Rounds:");

        psoRoundsSpinner.setModel(new javax.swing.SpinnerNumberModel(10, 10, 1000, 10));

        psoPopSizeLabel.setText("Swarm size:");

        psoPopSizeSpinner.setModel(new javax.swing.SpinnerNumberModel(50, 10, 1000, 10));

        javax.swing.GroupLayout psoPanelLayout = new javax.swing.GroupLayout(psoPanel);
        psoPanel.setLayout(psoPanelLayout);
        psoPanelLayout.setHorizontalGroup(
            psoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(psoPanelLayout.createSequentialGroup()
                .addContainerGap()
                .addComponent(c1Label)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c1Spinner, javax.swing.GroupLayout.PREFERRED_SIZE, 69,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(29, 29, 29)
                .addComponent(c2Label)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(c2Spinner, javax.swing.GroupLayout.PREFERRED_SIZE, 66,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(18, 18, 18)
                .addComponent(psoPopSizeLabel)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(psoPopSizeSpinner, javax.swing.GroupLayout.PREFERRED_SIZE, 57,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addGap(24, 24, 24)
                .addComponent(psoRoundsLabel)
                .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
                .addComponent(psoRoundsSpinner, javax.swing.GroupLayout.PREFERRED_SIZE, 63,
javax.swing.GroupLayout.PREFERRED_SIZE)
                .addContainerGap(javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
        );
        psoPanelLayout.setVerticalGroup(
            psoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
```

```
        .addGroup(psoPanelLayout.createSequentialGroup()
          .addContainerGap()
          .addGroup(psoPanelLayout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
            .addComponent(c1Label)
            .addComponent(c2Label)
            .addComponent(c1Spinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(c2Spinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(psoRoundsLabel)
            .addComponent(psoRoundsSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
            .addComponent(psoPopSizeLabel)
            .addComponent(psoPopSizeSpinner, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
          .addContainerGap(14, Short.MAX_VALUE))
      );

      javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
      getContentPane().setLayout(layout);
      layout.setHorizontalGroup(
        layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
        .addGroup(layout.createSequentialGroup()
          .addContainerGap()
          .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
            .addGroup(javax.swing.GroupLayout.Alignment.TRAILING, layout.createSequentialGroup()
              .addGap(0, 0, Short.MAX_VALUE)
              .addComponent(executeButton)
              .addGap(0, 0, Short.MAX_VALUE))
            .addComponent(resultsAreaScrollPane, javax.swing.GroupLayout.Alignment.TRAILING)
            .addGroup(layout.createSequentialGroup()
              .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
                .addComponent(trainingSetLabel)
                .addComponent(algorithmLabel))
              .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
              .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING,
false)
                .addComponent(trainingSetComboBox, 0, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE)
                .addComponent(algorithmComboBox, 0, javax.swing.GroupLayout.DEFAULT_SIZE,
Short.MAX_VALUE))
              .addGap(31, 31, 31)
              .addComponent(neuralNetworkPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
            .addComponent(bpPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
```

```
            .addComponent(gaPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addComponent(hgapsoPanel, javax.swing.GroupLayout.DEFAULT_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE)
            .addGroup(layout.createSequentialGroup()
              .addComponent(resultsLabel)
              .addGap(0, 0, Short.MAX_VALUE))
            .addComponent(psoPanel, javax.swing.GroupLayout.Alignment.TRAILING,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.DEFAULT_SIZE, Short.MAX_VALUE))
          .addContainerGap())
    );
    layout.setVerticalGroup(
      layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
      .addGroup(layout.createSequentialGroup()
        .addContainerGap()
        .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
          .addGroup(layout.createSequentialGroup()
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
              .addComponent(trainingSetLabel)
              .addComponent(trainingSetComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
            .addGap(18, 18, 18)
            .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.BASELINE)
              .addComponent(algorithmLabel)
              .addComponent(algorithmComboBox, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)))
          .addComponent(neuralNetworkPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE))
        .addGap(18, 18, 18)
        .addComponent(bpPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.UNRELATED)
        .addComponent(gaPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(psoPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(hgapsoPanel, javax.swing.GroupLayout.PREFERRED_SIZE,
javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.PREFERRED_SIZE)
        .addGap(18, 18, 18)
        .addComponent(executeButton)
        .addGap(4, 4, 4)
        .addComponent(resultsLabel)
        .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED)
```

```
            .addComponent(resultsAreaScrollPane, javax.swing.GroupLayout.DEFAULT_SIZE, 247,
Short.MAX_VALUE)
            .addContainerGap())
        );

        pack();
    }// </editor-fold>

    public void bpPrint(final BackPropagationAlgorithm bp, TrainingSet trainSet,
                TrainingSet testSet, NeuralNetwork network) {
        output.setText("");
        bp.setOutput(output);
        setVisible(true);
        SwingWorker<Void,Void> swingWorker = new SwingWorker<Void,Void>() {

            @Override
            protected Void doInBackground() throws Exception {
                output.append("Executing BackPropagation Algorithm\n");
                output.append("Neural Network Parameters:\n");
                output.append("   Number of layers: " + layersNoSpinner.getValue() + "\n");
                output.append("   Nodes per layer: " + nodesPerLayerSpinner.getValue() + "\n");
                output.append("Algorithm parameters:\n");
                output.append("   Epochs: " + bp.getMaxEpochs() + "\n");
                output.append("   Learning rate: " + bp.getLearningRate() + "\n");
                output.append("Executing");
                bp.execute(trainSet);
                output.append("\n\nFinished execution\n");
                ResultEvaluator eval = new ResultEvaluator();

                output.append("Mean square error: " +
                    df.format(eval.meanSquareError(network, testSet,0)) + "\n");
                return null;
            }
        };
        swingWorker.execute();
    }

    public void gaPrint(final GeneticAlgorithm ga, TrainingSet tSet) {
        output.setText("");
        ga.setOutput(output);
        setVisible(true);
        SwingWorker<Void,Void> swingWorker = new SwingWorker<Void,Void>() {

            @Override
```

```
        protected Void doInBackground() throws Exception {
          output.append("Executing Genetic Algorithm\n");
          output.append("Neural Network Parameters:\n");
          output.append("   Number of layers: " + layersNoSpinner.getValue() + "\n");
          output.append("   Nodes per layer: " + nodesPerLayerSpinner.getValue() + "\n");
          output.append("Algorithm parameters:\n");
          output.append("   Population size: " + (Integer) gaPopSizeSpinner.getValue() + "\n");
          output.append("   Rounds: " + ga.getMaxRounds() + "\n");
          output.append("   Crossover probability: " + ga.getPc() + "\n");
          output.append("   Mutation probability: " + ga.getPm() + "\n");
          output.append("Executing");
          ga.execute();
          NeuralNetwork network = ga.findBestAtom().getNeuralNetwork();
          output.append("\n\nFinished execution\n");
          ResultEvaluator eval = new ResultEvaluator();
          output.append("Mean square error: " +
               df.format(eval.meanSquareError(network, tSet,0)) + "\n");
          return null;
        }
      };
      swingWorker.execute();
    }

    public void psoPrint(final PSOAlgorithm pso, TrainingSet tSet) {
      output.setText("");
      pso.setOutput(output);
      setVisible(true);
      SwingWorker<Void,Void> swingWorker = new SwingWorker<Void,Void>() {

        @Override
        protected Void doInBackground() throws Exception {
          output.append("Executing Particle Swarm Optimization\n");
          output.append("Neural Network Parameters:\n");
          output.append("   Number of layers: " + layersNoSpinner.getValue() + "\n");
          output.append("   Nodes per layer: " + nodesPerLayerSpinner.getValue() + "\n");
          output.append("Algorithm parameters:\n");
          output.append("   Population size: " + (Integer) psoPopSizeSpinner.getValue() + "\n");
          output.append("   Rounds: " + pso.getMaxRounds() + "\n");
          output.append("   C1: " + pso.getC1() + "\n");
          output.append("   C2 " + pso.getC2() + "\n");
          output.append("Executing");
          pso.execute();
          NeuralNetwork network = pso.findBestAtom().getNeuralNetwork();
          output.append("\n\nFinished execution\n");
```

```
            ResultEvaluator eval = new ResultEvaluator();
            output.append("Mean square error: " +
                  df.format(eval.meanSquareError(network, tSet,0)) + "\n");
            return null;
          }
        };
        swingWorker.execute();
    }

    public void hgapsoPrint(final HGAPSOAlgorithm hgapso, TrainingSet tSet) {
        output.setText("");
        hgapso.setOutput(output);
        setVisible(true);
        SwingWorker<Void,Void> swingWorker = new SwingWorker<Void,Void>() {

          @Override
          protected Void doInBackground() throws Exception {
            output.append("Executing Hybrid Genetic Algorithm Particle Swarm Optimization\n");
            output.append("Neural Network Parameters:\n");
            output.append("   Number of layers: " + layersNoSpinner.getValue() + "\n");
            output.append("   Nodes per layer: " + nodesPerLayerSpinner.getValue() + "\n");
            output.append("Hybrid Genetic Algorithm Particle Swarm parameters:\n");
            output.append("   Population size: " + (Integer) hgapsoPopSizeSpinner.getValue() + "\n");
            output.append("   Rounds: " + hgapso.getMaxRounds() + "\n");
            output.append("   Crossover probability: " + hgapso.getPc() + "\n");
            output.append("   Mutation probability: " + hgapso.getPm() + "\n");
            output.append("   C1: " + hgapso.getC1() + "\n");
            output.append("   C2 " + hgapso.getC2() + "\n");
            output.append("Executing");
            hgapso.execute();
            NeuralNetwork network = hgapso.getBestNetwork();
            output.append("\n\nFinished execution\n");
            ResultEvaluator eval = new ResultEvaluator();
            output.append("Mean square error: " +
                  df.format(eval.meanSquareError(network, tSet,0)) + "\n");
            return null;
          }
        };
        swingWorker.execute();
    }

    private void executeButtonActionPerformed(java.awt.event.ActionEvent evt) {

        int algorithmIndex = algorithmComboBox.getSelectedIndex();
```

```
int trainingSetIndex = trainingSetComboBox.getSelectedIndex();

int layersNo = (Integer) layersNoSpinner.getValue();
int nodesPerLayerNo = (Integer) nodesPerLayerSpinner.getValue();

TrainingSet tSet = new TrainingSet();
tSet.readXMLFile(trainingSetFiles.get(trainingSetIndex).getAbsolutePath());
NeuralNetwork network;
if (algorithmIndex == 0) {
   int epochs = (Integer) bpEpochsSpinner.getValue();
   network = new NeuralNetwork();
   network.createNetworkByTrainingSet(tSet, layersNo, nodesPerLayerNo);
   BackPropagationAlgorithm bp = new BackPropagationAlgorithm(network,epochs);
   this.bpPrint(bp,tSet.trainSet(1),tSet.testSet(1),network);
} else if (algorithmIndex == 1) {
   double pc = (Double) pcSpinner.getValue();
   double pm = (Double) pmSpinner.getValue();
   int popSize = (Integer) gaPopSizeSpinner.getValue();
   int rounds = (Integer) gaRoundsSpinner.getValue();
   GeneticAlgorithm ga = new GeneticAlgorithm(popSize, rounds, pc, pm,
         trainingSetFiles.get(trainingSetIndex).getAbsolutePath(),
         layersNo, nodesPerLayerNo);
   this.gaPrint(ga,tSet);
} else if (algorithmIndex == 2) {
   double c1 = (Double) c1Spinner.getValue();
   double c2 = (Double) c2Spinner.getValue();
   int popSize = (Integer) psoPopSizeSpinner.getValue();
   int rounds =(Integer) psoRoundsSpinner.getValue();
   PSOAlgorithm pso = new PSOAlgorithm(popSize, rounds, c1, c2,
         trainingSetFiles.get(trainingSetIndex).getAbsolutePath(),
         layersNo, nodesPerLayerNo);
   this.psoPrint(pso,tSet);
} else if (algorithmIndex == 3) {
   double pc = (Double) pcSpinner.getValue();
   double pm = (Double) pmSpinner.getValue();
   double c1 = (Double) c1Spinner.getValue();
   double c2 = (Double) c2Spinner.getValue();
   int popSize = (Integer) hgapsoPopSizeSpinner.getValue();
   int rounds = (Integer) hgapsoRoundsSpinner.getValue();
   HGAPSOAlgorithm hgapso = new HGAPSOAlgorithm(popSize, rounds, pc, pm,
         c1, c2, trainingSetFiles.get(trainingSetIndex).getAbsolutePath(),
         layersNo, nodesPerLayerNo);
   this.hgapsoPrint(hgapso,tSet);
}
```

```
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String args[]) {
        /* Set the Nimbus look and feel */
        //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (optional) ">
        /* If Nimbus (introduced in Java SE 6) is not available, stay with the default look and feel.
         * For details see http://download.oracle.com/javase/tutorial/uiswing/lookandfeel/plaf.html
         */
        try {
            for (javax.swing.UIManager.LookAndFeelInfo info :
javax.swing.UIManager.getInstalledLookAndFeels()) {
                if ("Nimbus".equals(info.getName())) {
                    javax.swing.UIManager.setLookAndFeel(info.getClassName());
                    break;
                }
            }
        } catch (ClassNotFoundException ex) {

java.util.logging.Logger.getLogger(TrainingSetSelector.class.getName()).log(java.util.logging.Level.SEVER
E, null, ex);
        } catch (InstantiationException ex) {

java.util.logging.Logger.getLogger(TrainingSetSelector.class.getName()).log(java.util.logging.Level.SEVER
E, null, ex);
        } catch (IllegalAccessException ex) {

java.util.logging.Logger.getLogger(TrainingSetSelector.class.getName()).log(java.util.logging.Level.SEVER
E, null, ex);
        } catch (javax.swing.UnsupportedLookAndFeelException ex) {

java.util.logging.Logger.getLogger(TrainingSetSelector.class.getName()).log(java.util.logging.Level.SEVER
E, null, ex);
        }
        //</editor-fold>

        /* Create and display the form */
        java.awt.EventQueue.invokeLater(new Runnable() {
            public void run() {
                new TrainingSetSelector().setVisible(true);
            }
        });
    }
```

```
    // Variables declaration - do not modify
    private javax.swing.JComboBox algorithmComboBox;
    private javax.swing.JLabel algorithmLabel;
    private javax.swing.JLabel bpEpochsLabel;
    private javax.swing.JSpinner bpEpochsSpinner;
    private javax.swing.JPanel bpPanel;
    private javax.swing.JLabel c1Label;
    private javax.swing.JSpinner c1Spinner;
    private javax.swing.JLabel c2Label;
    private javax.swing.JSpinner c2Spinner;
    private javax.swing.JButton executeButton;
    private javax.swing.JPanel gaPanel;
    private javax.swing.JLabel gaPopSizeLabel;
    private javax.swing.JSpinner gaPopSizeSpinner;
    private javax.swing.JLabel gaRoundsLabel;
    private javax.swing.JSpinner gaRoundsSpinner;
    private javax.swing.JPanel hgapsoPanel;
    private javax.swing.JLabel hgapsoPopSizeLabel;
    private javax.swing.JSpinner hgapsoPopSizeSpinner;
    private javax.swing.JLabel hgapsoRoundsLabel;
    private javax.swing.JSpinner hgapsoRoundsSpinner;
    private javax.swing.JLabel layersNoLabel;
    private javax.swing.JSpinner layersNoSpinner;
    private javax.swing.JPanel neuralNetworkPanel;
    private javax.swing.JLabel nodesPerLayerLabel;
    private javax.swing.JSpinner nodesPerLayerSpinner;
    private javax.swing.JTextArea output;
    private javax.swing.JLabel pcLabel;
    private javax.swing.JSpinner pcSpinner;
    private javax.swing.JLabel pmLabel;
    private javax.swing.JSpinner pmSpinner;
    private javax.swing.JPanel psoPanel;
    private javax.swing.JLabel psoPopSizeLabel;
    private javax.swing.JSpinner psoPopSizeSpinner;
    private javax.swing.JLabel psoRoundsLabel;
    private javax.swing.JSpinner psoRoundsSpinner;
    private javax.swing.JScrollPane resultsAreaScrollPane;
    private javax.swing.JLabel resultsLabel;
    private javax.swing.JComboBox trainingSetComboBox;
    private javax.swing.JLabel trainingSetLabel;
    // End of variables declaration
}
```

```
// Στο αρχείο αυτό ορίζονται όλες οι απαραίτητες μέθοδοι για την μετατροπή
// ενός διανύσματος πραγματικών τιμών σε μία δυαδική συμβολοσειρά.

package hgapso;

import java.util.ArrayList;

public class BinaryArithmetic {

   // Μετατρέπει έναν ακέραιο αριθμό x από το δεκαδικό σύστημα αρίθμησης
   // στο δυαδικό. Τα bits του δυαδικού αριθμού επιστρέφονται μέσα σε έναν
   // πίνακα ακεραίων.
   public static int[] decToBinary(int x) {

      ArrayList<Integer> tmp = new ArrayList<Integer>();
      if (x == 0)
         tmp.add(0);
      else {
         while (x > 0) {
            tmp.add(x % 2);
            x /= 2;
         }
      }
      int[] bin = new int[tmp.size()];
      for (int i = tmp.size()-1; i >= 0; i--) {
          bin[tmp.size()-1-i] = tmp.get(i);
      }
      return bin;

   }

   // Μετατρέπει στην αντίστοιχη δεκαδική αναπαράσταση, τον δυαδικό αριθμό
   // που υπάρχει σε έναν πίνακα array, από την θέση from ως την θέση to
   public static int binaryToDec(int[] array, int from, int to) {

      int dec = 0;
      int d = to-from;
      for (int i = from; i <= to; i++) {
          if (array[i] == 1) {
             dec += Math.pow(2,d);
          }
          d--;
      }
      return dec;
```

```
  }

  // Μετατρέπει στην αντίστοιχη δεκαδική αναπαράσταση, τον δυαδικό αριθμό
  // που υπάρχει σε έναν πίνακα array
  public static int binaryToDec(int[] array) {

     return binaryToDec(array,0,array.length-1);

  }

  // Υπολογίζει και επιστρέφει τον αριθμό των απαιτούμενων bits τα οποία είναι
  // απαραίτητα για την κωδικοποίηση πραγματικών τιμών στο διάστημα [min,max]
  // με ακρίβεια a δεκαφικών ψηφίων
  public static int findBitsNum(double min, double max, int a) {

     int m;
     for (m = 1; ; m++) {
        if ((max-min)*Math.pow(10,a) <= Math.pow(2,m)-1) {
           break;
        }
     }
     return m;

  }

  // Μετατρέπει μία πραγματική τιμή x στο δυαδικό σύστημα αρίθμησης. Η
  // τιμή κινείται μεταξύ των ορίων [min,max]. Ο αριθμός ο οποίος επιστρέφεται
  // αποτελείται από m bits.
  public static int[] doubleToBinary(double x, double min, double max, int m) {

     // Μετατροπή της πραγματικής τιμής σε ακέραια τιμή (d)
     int d = (int) Math.round((x-min)*((Math.pow(2,m)-1)/(max-min)));
     // Μετατροπή του d στο δυαδικό σύστημα αρίθμησης
     int[] tmp = decToBinary(d);
     // Έλεγξε αν ο δυαδικός αριθμός αποτελείται από m bits
     if (tmp.length < m) {
        // Αν όχι, τότε συμπλήρωσέ τον με μηδενικά στην αρχή
        int[] bin = new int[m];
        for (int i = 0; i < tmp.length; i++) {
           bin[m-tmp.length+i] = tmp[i];
        }
        return bin;
     } else {
```

```
            return tmp;
        }

    }

    // Μετατρέπει ένα πραγματικό διάνυσμα τιμών στην δυαδική του αναπαράσταση.
    // Κάθε τιμή x[i] κινείται μεταξύ των ορίων [min[i],max[i] και αναπαρίσταται
    // με m[i] bits.
    public static int[] doubleToBinary(double[] x, double[] min, double[] max, int[] m) {

        int totalBits = 0;
        for (int mi : m) {
            totalBits += mi;
        }
        int[] bin = new int[totalBits];
        int k = 0;
        for (int i = 0; i < x.length; i++) {
            int[] binXi = doubleToBinary(x[i],min[i],max[i],m[i]);
            for (int j = 0; j < binXi.length; j++) {
                bin[k++] = binXi[j];
            }
        }
        return bin;

    }

    // Μετατρέπει έναν αριθμό στο δυαδικό ο οποίος αντιστοιχεί σε ένα πραγματικό
    // διάνυσμα στο αντίστοιχο διάνυσμα (x). Κάθε τιμή x[i] κινείται μεταξύ των
    // ορίων [min[i],max[i] και αναπαρίσταται με m[i] bits.
    public static double[] binaryToDouble(int[] bin, double[] min, double[] max, int[] m) {

        double[] x = new double[m.length];
        int from = 0;
        for (int i = 0; i < x.length; i++) {
            int to = from + m[i]-1;
            int d = binaryToDec(bin,from,to);
            x[i] = Math.round(min[i] + d*((max[i]-min[i])/(Math.pow(2,m[i])-1)));
            from = to+1;
        }
        return x;

    }

}
```

```
package hgapso;

import neuralnetwork.NeuralNetwork;

public class GAGenotype extends Genotype {

   private int[] m;
   private int[] bin;
   private double[] min;
   private double[] max;
   private double selectionProb;
   private double accumulativeProb;

   public GAGenotype(String id) {
      super(id);
   }

   public GAGenotype(String id, NeuralNetwork neuralNetwork) {
      super(id, neuralNetwork);
   }

   public double getSelectionProb() {
      return selectionProb;
   }

   public void setSelectionProb(double selectionProb) {
      this.selectionProb = selectionProb;
   }

   public double getAccumulativeProb() {
      return accumulativeProb;
   }

   public void setAccumulativeProb(double accumulativeProb) {
      this.accumulativeProb = accumulativeProb;
   }

   public int getBit(int i) {
      return bin[i];
   }

   public void setBit(int i, int bit) {
      bin[i] = bit;
   }
```

```
  public GAGenotype(GAGenotype g) {
    super(g.id, g.neuralNetwork);
    this.genes = new double[g.genes.length];
    for (int i = 0; i < genes.length; i++)
        this.genes[i] = g.genes[i];
    this.eval = g.eval;
    this.selectionProb = g.selectionProb;
    this.accumulativeProb = g.accumulativeProb;
    this.m = new int[g.m.length];
    for (int i = 0; i < m.length; i++)
        this.m[i] = g.m[i];
    this.bin = new int[g.bin.length];
    for (int i = 0; i < bin.length; i++)
        this.bin[i] = g.bin[i];
    this.max = new double[g.max.length];
    for (int i = 0; i < max.length; i++)
        this.max[i] = g.max[i];
    this.min = new double[g.min.length];
    for (int i = 0; i < min.length; i++)
        this.min[i] = g.min[i];
  }

  public int getGenotypeSize() {

    int totalBits = 0;
    for (int i = 0; i < m.length; i++)
        totalBits += m[i];
    return totalBits;

  }

  public void calcBitNum() {

    m = new int[genes.length];
    min = new double[genes.length];
    max = new double[genes.length];
    for (int i = 0; i < genes.length; i++) {
        min[i] = GeneticAlgorithm.MIN_INIT_VAL;
        max[i] = GeneticAlgorithm.MAX_INIT_VAL;
    }
    for (int i = 0; i < genes.length; i++) {
        m[i] = BinaryArithmetic.findBitsNum(GeneticAlgorithm.MIN_INIT_VAL,
                                GeneticAlgorithm.MAX_INIT_VAL,
```

```
                                GeneticAlgorithm.ACCURACY);
        }

    }

    public void encode() {

        bin = BinaryArithmetic.doubleToBinary(genes,min,max,m);

    }

    public void decode() {

        genes = BinaryArithmetic.binaryToDouble(bin,min,max,m);

    }

    public void mutateBit(int i) {

        if (bin[i] == 0)
            bin[i] = 1;
        else
            bin[i] = 0;

    }

    @Override
    public String toString() {

        return id + ": " + this.getEval();

    }

}
```

```
package hgapso;

import neuralnetwork.NeuralNetwork;
import neuralnetwork.TrainingSet;

public class GAPopulation extends Population {

   private String networkXMLFile;
   private TrainingSet tSet;
   private int hiddenLayersNo;
   private int nodesPerLayer;

   // Constructors
   public GAPopulation(int popSize, String networkXMLFile) {
      this.popSize = popSize;
      this.networkXMLFile = networkXMLFile;
   }

   public GAPopulation(int popSize, TrainingSet tSet, int hiddenLayersNo, int nodesPerLayer) {
      this.popSize = popSize;
      this.tSet = tSet;
      this.hiddenLayersNo = hiddenLayersNo;
      this.nodesPerLayer = nodesPerLayer;
   }

   @Override
   public void initializePopulation() {
      atoms = new Genotype[popSize];
      for (int i = 0; i < atoms.length; i++) {
         NeuralNetwork network = new NeuralNetwork();
         if (networkXMLFile != null)
            network.readXMLFile(networkXMLFile);
         else
            network.createNetworkByTrainingSet(tSet,hiddenLayersNo,nodesPerLayer);

network.randomInitWeights(GeneticAlgorithm.MIN_INIT_VAL,GeneticAlgorithm.MAX_INIT_VAL);
         atoms[i] = new GAGenotype("ga" + (i+1));
         atoms[i].setNeuralNetwork(network);
         atoms[i].init();
         ((GAGenotype) atoms[i]).calcBitNum();
         ((GAGenotype) atoms[i]).encode();
      }
   }
```

```
    @Override
    public void updateWeights() {

        for (Genotype a : atoms)
            ((GAGenotype) a).updateWeights();

    }

}
```

```
package hgapso;

import java.util.ArrayList;
import javax.swing.JTextArea;
import neuralnetwork.TrainingSet;

public class GeneticAlgorithm {

   public static final double MIN_INIT_VAL = -10;
   public static final double MAX_INIT_VAL = +10;
   public static final int ACCURACY = 5;

   private GAPopulation pop;

   private int popSize;
   private int rounds;
   private int maxRounds;
   private double pc;
   private double pm;
   private String networkXMLFile;
   private String trainingSetXMLFile;
   private int hiddenLayersNo;
   private int nodesPerLayer;

   private TrainingSet tSet;

   private JTextArea output;

   public GeneticAlgorithm(int popSize, int maxRounds, double pc, double pm,
                   String networkXMLFile, String trainingSetXMLFile) {
      this.popSize = popSize;
      this.maxRounds = maxRounds;
      this.pc = pc;
      this.pm = pm;
      this.networkXMLFile = networkXMLFile;
      this.trainingSetXMLFile = trainingSetXMLFile;
   }

   public GeneticAlgorithm(int popSize, int maxRounds, double pc, double pm,
                    String trainingSetXMLFile, int hiddenLayersNo, int nodesPerLayer) {
      this.popSize = popSize;
      this.maxRounds = maxRounds;
      this.pc = pc;
      this.pm = pm;
```

```
    this.trainingSetXMLFile = trainingSetXMLFile;
    this.hiddenLayersNo = hiddenLayersNo;
    this.nodesPerLayer = nodesPerLayer;
  }

  public void initialize() {

    tSet = new TrainingSet();
    tSet.readXMLFile(trainingSetXMLFile);
    if (networkXMLFile != null)
      pop = new GAPopulation(popSize,networkXMLFile);
    else
      pop = new GAPopulation(popSize,tSet,hiddenLayersNo,nodesPerLayer);
    pop.initializePopulation();

  }

  public void evaluate() {

    // Βαθμολόγησε το κάθε άτομο του πληθυσμού
    for (int i = 0; i < pop.getPopSize(); i++) {
       double eval = pop.getAtom(i).evaluate(tSet);
    }

    // Θα πρέπει να διασφαλίσουμε ότι κάθε άτομο
    // δεν έχει βαθμολογία 0
    // Βρες το minimum evaluation στον πληθυσμό
    int k = 0;
    double minEval = 0.0;
    for (int i = 0; i < pop.getPopSize(); i++) {
      if (pop.getAtom(i).getEval() > 0) {
        minEval = pop.getAtom(i).getEval();
        k = i;
        break;
      }
    }

    for (int i = k+1; i < pop.getPopSize(); i++) {
      if (pop.getAtom(i).getEval() > 0 &&
        pop.getAtom(i).getEval() < minEval)
        minEval = pop.getAtom(i).getEval();
    }
    // Αν το minEval είναι 0 (όλα μηδέν) τότε θέσε το minEval ίσο με 10
    if (minEval == 0)
```

```
        minEval = 10;

    // Για κάθε άτομο του πληθυσμού που έχει evaluation 0, θέσε την
    // τιμή αξιολόγησής του ίση με minEval/10
    for (int i = 0; i < pop.getPopSize(); i++) {
        if (pop.getAtom(i).getEval() == 0)
            pop.getAtom(i).setEval(minEval/10);
    }

  }

  public void select() {

    // Βρες την συνολική καταλληλότητα του πληθυσμού
    double totalEval = 0;
    for (int i = 0; i < pop.getPopSize(); i++) {
        totalEval += pop.getAtom(i).getEval();
    }
    // Υπολόγισε την πιθανότητα επιλογής
    for (int i = 0; i < pop.getPopSize(); i++) {
        double selectionProb = pop.getAtom(i).getEval() / totalEval;
        ((GAGenotype)pop.getAtom(i)).setSelectionProb(selectionProb);
    }
    // Υπολόγησε την συσσωρευμένη πιθανότητα επιλογής για κάθε άτομο
    double q =  ((GAGenotype)pop.getAtom(0)).getSelectionProb();
   ((GAGenotype) pop.getAtom(0)).setAccumulativeProb(q);
    for (int i = 1; i < pop.getPopSize(); i++) {
        q += ((GAGenotype)pop.getAtom(i)).getSelectionProb();
        ((GAGenotype)pop.getAtom(i)).setAccumulativeProb(q);
    }
    // Δημιούργησε το νέο πληθυσμό
    GAPopulation selectionPop;
    if (networkXMLFile != null)
       selectionPop = new GAPopulation(popSize,networkXMLFile);
    else
       selectionPop = new GAPopulation(popSize,tSet,hiddenLayersNo,nodesPerLayer);
    selectionPop.initializePopulation();
    // Ο νέος πληθυσμός έχει το ίδιο μέγεθος με τον παλιό (popSize)
    for (int i = 0; i < pop.getPopSize(); i++) {
        // Δημιούργησε έναν τυχαίο αριθμό r: 0 < r < 1
        double r = Utils.randValue();
        GAGenotype selectedAtom = null;
        // Αν r <= q0
        if (r <= ((GAGenotype)pop.getAtom(0)).getAccumulativeProb()) {
```

```
            // τότε το πρώτο άτομο (θέση 0) επιλέγεται
            selectedAtom = (GAGenotype) pop.getAtom(0);
        } else {
            // σε διαφορετική περίπτωση επιλέγεται το άτομο j τέτοιο ώστε: qj-1 < r <= qj
            for (int j = 1; j < pop.getPopSize(); j++) {
                if (((GAGenotype)pop.getAtom(j-1)).getAccumulativeProb() < r &&
                    r <= ((GAGenotype)pop.getAtom(j)).getAccumulativeProb()) {
                    selectedAtom = (GAGenotype) pop.getAtom(j);
                }
            }
        }
        // Κλωνοποίησε το επιλεγμένο άτομο με την χρήση του Copy Constructor
        selectedAtom = new GAGenotype(selectedAtom);
        // Πρόσθεσέ το στο νέο πληθυσμό
        selectionPop.setAtom(i, selectedAtom);
        // Άλλαξε το id του
        selectionPop.getAtom(i).setId("ga" + (i+1));
    }
    // Θέσε το νέο πληθυσμό ως νέο
    pop = selectionPop;
}

private int findBestAtomPos() {

    int pos = -1;
    double maxEval = 0;
    for (int i = 0; i < pop.getPopSize(); i++) {
        if (pop.getAtom(i) != null && pop.getAtom(i).getEval() > maxEval) {
            maxEval = pop.getAtom(i).getEval();
            pos = i;
        }
    }
    return pos;

}

// Διασταύρωση μονού σημείου
private void singlePointCrossOver(GAGenotype gen1, GAGenotype gen2) {
    // Βρες τα πιθανά m-1 σημεία κοπής
    int m = gen1.getGenotypeSize()-1;
    // Βρες την πιθανότητα κοπής του καθενός (ισοπίθανα σημεία κοπής)
    double p = 1 / (double) m;
    // Βρες το σημείο κοπής
    // Επέλεξε αρχικά έναν τυχαίο αριθμό r: 0 < r < 1
```

```
        double r = Utils.randValue();
        // Το σημείο κοπής
        int k;
        // Αν r < p, επέλεξε το πρώτο σημείο
        if (r < p)
           k = 1;
        else {
           // Αλλιώς επέλεξε το σημείο για το οποίο ισχύει ότι: (k-1)*p < r <= k*p
           for (k = 2; k <= m; k++) {
               if ((k-1)*p < r && r <= k*p)
                  break;
           }
        }
        // Αντέστρεψε τα γονίδια από την θέση k και μετά
        for (int i = k; i <= m; i++) {
            int tmp = gen1.getBit(i);
            gen1.setBit(i, gen2.getBit(i));
            gen2.setBit(i, tmp);
        }
    }

    public void crossover() {

        // Επιλογή των ατόμων που θα διασταυρωθούν
        ArrayList<GAGenotype> newGenotypes = new ArrayList<>();
        // Για κάθε άτομο του πληθυσμού
        for (int i = 0; i < pop.getPopSize(); i++) {
           // Έλεγξε αν το άτομο θα διασταυρωθεί
           // Επέλεξε αρχικά έναν τυχαίο αριθμό r: 0 < r < 1
           double r = Utils.randValue();
           // Αν ο αριθμός είναι μικρότερος της πιθανότητας διασταύρωσης
           if (r < pc) {
              // Το άτομο επιλέγεται για διασταύρωση
              newGenotypes.add((GAGenotype)pop.getAtom(i));
              pop.setAtom(i, null);
           }
        }
        // Αν ο αριθμός των ατόμων είναι μονός
        if (newGenotypes.size() % 2 == 1) {
           // Πρόσθεσε για διασταύρωση το καλύτερο άτομο του πληθυσμού
           // από τα εναπομείναντα, μη επιλεχθέντα άτομα του πληθυσμού
           // Βρες αρχικά την θέση στην οποία αυτό βρίσκεται
           int bestAtomPos = findBestAtomPos();
           if (bestAtomPos == -1) {
```

```
            // Αυτό σημαίνει ότι το μέγεθος του πληθυσμού είναι μονό και ότι
            // όλα έχουν επιλεχθεί για διασταύρωση! Συνεπώς αφαίρεσε ένα άτομο
            // από τα επιλεχθέντα προς διασταύρωση
            // στην περίπτωση αυτή (υποθέτουμε το πρώτο)
            pop.setAtom(0, newGenotypes.get(0));
            newGenotypes.remove(0);
          } else {
            // Βάλε το άτομο μέσα στην λίστα των επιλεχθέντων
            newGenotypes.add((GAGenotype)pop.getAtom(bestAtomPos));
            pop.setAtom(bestAtomPos, null);
          }
        }
        // Για κάθε ζεύγος που επιλέχθηκε για διασταύρωση
        for (int i = 0; i < newGenotypes.size(); i += 2) {
          // Διασταύρωσέ το
          singlePointCrossOver(newGenotypes.get(i),newGenotypes.get(i+1));
        }
        // Ανανέωσε τον πληθυσμό
        for (int i = 0; i < pop.getPopSize(); i++) {
          if (pop.getAtom(i) == null) {
            pop.setAtom(i, newGenotypes.get(0));
            pop.getAtom(i).setId("ga" + (i+1));
            newGenotypes.remove(0);
          }
        }
      }

      public void mutate() {

        // Για κάθε άτομο του πληθυσμού
        for (int i = 0; i < pop.getPopSize(); i++) {
          // Για κάθε γονίδιο του ατόμου
          for (int j = 0; j < ((GAGenotype) pop.getAtom(i)).getGenotypeSize(); j++) {
            // Επέλεξε έναν τυχαίο αριθμό r: 0 < r < 1
            double r = Utils.randValue();
            // Αν ισχύει ότι r < pm, το γονίδιο μεταλλάσεται
            if (r < pm) {
              ((GAGenotype) pop.getAtom(i)).mutateBit(j);
            }
          }
        }

      }
```

```
  public boolean endCriteria() {
     return (rounds == maxRounds);
  }

  public void setOutput(JTextArea output) {
     this.output = output;
  }

  public void execute() {

     rounds = 0;
     this.initialize();
     while (!endCriteria()) {
          this.evaluate();
          this.select();
          this.crossover();
          this.mutate();
          rounds++;
          if (rounds % 10 == 0) {
             output.append(".");
          }
          pop.updateWeights();
     }
     this.evaluate();
     int bestAtomPos = findBestAtomPos();
     GAGenotype best = (GAGenotype) pop.getAtom(bestAtomPos);

  }

  public void print() {

     pop.printPopulation();

  }

  public GAGenotype findBestAtom() {

     return (GAGenotype) pop.getAtom(this.findBestAtomPos());

  }

  public GAPopulation getPop() {
     return pop;
  }
```

```
    public int getMaxRounds() {
        return maxRounds;
    }

    public double getPc() {
        return pc;
    }

    public double getPm() {
        return pm;
    }

}
```

```
package hgapso;

import neuralnetwork.Connection;
import neuralnetwork.InputNeuron;
import neuralnetwork.NeuralNetwork;
import neuralnetwork.Neuron;
import neuralnetwork.TrainingSet;

public abstract class Genotype {

   protected String id;
   protected NeuralNetwork neuralNetwork;
   protected double[] genes;
   protected double eval;

   public Genotype(String id) {
      this.id = id;
   }

   public Genotype(String id, NeuralNetwork neuralNetwork) {
      this.id = id;
      this.neuralNetwork = neuralNetwork;
   }

   public String getId() {
      return id;
   }

   public void setId(String id) {
      this.id = id;
   }

   public NeuralNetwork getNeuralNetwork() {
      return neuralNetwork;
   }

   public void setNeuralNetwork(NeuralNetwork neuralNetwork) {
      this.neuralNetwork = neuralNetwork;
   }

   public double[] getGenes() {
      return genes;
   }
```

```
    public double getEval() {
        return eval;
    }

    public void setEval(double eval) {
        this.eval = eval;
    }

    public void init() {

        genes = new double[neuralNetwork.numberOfWeights()];
        int i = 0;
        for (Neuron n : neuralNetwork.getNeurons()) {
            for (Connection conn : n.getOutgoingConnections()) {
                genes[i++] = conn.getWeight();
            }
            if (!(n instanceof InputNeuron))
                genes[i++] = n.getThreshold();
        }

    }

    public double evaluate(TrainingSet tSet) {

        double[][] inputs = tSet.getInputs();
        double[][] outputs = tSet.getOutputs();

        eval = (1 / neuralNetwork.squareError(inputs,outputs)) * 100;
        return eval;

    }

    public void updateWeights() {
        int i = 0;
        for (Neuron n : neuralNetwork.getNeurons()) {
            for (Connection conn : n.getOutgoingConnections()) {
                conn.setWeight(genes[i++]);
            }
            if (!(n instanceof InputNeuron))
                n.setThreshold(genes[i++]);
        }

    }
}
```

```
package hgapso;

import javax.swing.JTextArea;
import neuralnetwork.NeuralNetwork;

public class HGAPSOAlgorithm {

   private int popSize;
   private int maxRounds;
   private double pc;
   private double pm;
   private double c1;
   private double c2;
   private String networkXMLFile;
   private String trainingSetXMLFile;
   private int hiddenLayersNo;
   private int nodesPerLayer;

   private GeneticAlgorithm ga;
   private PSOAlgorithm pso;

   private JTextArea output;

   public HGAPSOAlgorithm(int popSize, int maxRounds, double pc, double pm,
                  double c1, double c2,
                  String networkXMLFile, String trainingSetXMLFile) {
      this.popSize = popSize;
      this.maxRounds = maxRounds;
      this.pc = pc;
      this.pm = pm;
      this.c1 = c1;
      this.c2 = c2;
      this.networkXMLFile = networkXMLFile;
      this.trainingSetXMLFile = trainingSetXMLFile;
      ga = new GeneticAlgorithm(popSize, maxRounds / 80, pc, pm, networkXMLFile, trainingSetXMLFile);
      pso = new PSOAlgorithm(popSize, maxRounds / 20, c1, c2, networkXMLFile, trainingSetXMLFile);
   }

   public HGAPSOAlgorithm(int popSize, int maxRounds, double pc, double pm,
                  double c1, double c2,
                  String trainingSetXMLFile,
                  int hiddenLayersNo, int nodesPerLayer) {
      this.popSize = popSize;
      this.maxRounds = maxRounds;
```

```
        this.pc = pc;
        this.pm = pm;
        this.c1 = c1;
        this.c2 = c2;
        this.trainingSetXMLFile = trainingSetXMLFile;
        this.hiddenLayersNo = hiddenLayersNo;
        this.nodesPerLayer = nodesPerLayer;
        ga = new GeneticAlgorithm(popSize, (int) (maxRounds * 0.3), pc, pm, trainingSetXMLFile, hiddenLayersNo, nodesPerLayer);
        pso = new PSOAlgorithm(popSize, (int) (maxRounds * 0.7), c1, c2, trainingSetXMLFile, hiddenLayersNo, nodesPerLayer);
    }

    public void execute() {

        ga.execute();

        pso.execute(ga.getPop());

    }

    public NeuralNetwork getBestNetwork() {

        return pso.findBestAtom().getNeuralNetwork();

    }

    public void setOutput(JTextArea output) {
        ga.setOutput(output);
        pso.setOutput(output);
        this.output = output;
    }

    public int getMaxRounds() {
        return maxRounds;
    }

    public double getPc() {
        return pc;
    }

    public double getPm() {
        return pm;
    }
```

```
    public double getC1() {
        return c1;
    }

    public double getC2() {
        return c2;
    }



}
```

```
package hgapso;

import javax.swing.JTextArea;
import neuralnetwork.TrainingSet;

public class PSOAlgorithm {

   // Ελάχιστη και μέγιστη τιμή αρχικοποίησης του κάθε στοιχείου κάθε
   // μορίου του σμήνους
   public static final double MIN_INIT_VAL = -10;
   public static final double MAX_INIT_VAL = +10;

   private PSOPopulation pop;

   private int popSize;
   private int maxRounds;
   private double c1;
   private double c2;
   private String networkXMLFile;
   private String trainingSetXMLFile;

   private TrainingSet tSet;
   private int hiddenLayersNo;
   private int nodesPerLayer;

   private JTextArea output;

   public PSOAlgorithm(int popSize, int maxRounds, double c1, double c2,
            String networkXMLFile, String trainingSetXMLFile) {
      this.popSize = popSize;
      this.maxRounds = maxRounds;
      this.c1 = c1;
      this.c2 = c2;
      this.networkXMLFile = networkXMLFile;
      this.trainingSetXMLFile = trainingSetXMLFile;
   }

   public PSOAlgorithm(int popSize, int maxRounds, double c1, double c2,
            String trainingSetXMLFile, int hiddenLayersNo, int nodesPerLayer) {
      this.popSize = popSize;
      this.maxRounds = maxRounds;
      this.c1 = c1;
      this.c2 = c2;
      this.trainingSetXMLFile = trainingSetXMLFile;
```

```
    this.hiddenLayersNo = hiddenLayersNo;
    this.nodesPerLayer = nodesPerLayer;
  }

  public void initialize() {

    tSet = new TrainingSet();
    tSet.readXMLFile(trainingSetXMLFile);
    if (networkXMLFile != null)
      pop = new PSOPopulation(popSize,c1,c2,networkXMLFile);
    else
      pop = new PSOPopulation(popSize,c1,c2,tSet,hiddenLayersNo,nodesPerLayer);
    pop.initializePopulation();

  }

  public void initialize(GAPopulation gapop) {

    pop = new PSOPopulation(popSize,c1,c2,networkXMLFile);
    pop.initializePopulation(gapop);
    tSet = new TrainingSet();
    tSet.readXMLFile(trainingSetXMLFile);

  }

  public void setPop(PSOPopulation pop) {
    this.pop = pop;
  }

  public void settSet(TrainingSet tSet) {
    this.tSet = tSet;
  }

  public void execute() {

    // Αρχικοποίηση του σμήνους στον χώρο αναζήτησης
    this.initialize();

    // Εξέλιξη του σμήνους
    this.evolution();

  }

  public void execute(GAPopulation gapop) {
```

```
    // Αρχικοποίηση του σμήνους στον χώρο αναζήτησης
    this.initialize(gapop);

    // Εξέλιξη του σμήνους
    this.evolution();

  }

  // Εξέλιξη του σμήνους
  public void evolution() {

    // Αξιολόγηση του αρχικού πληθυσμού
    pop.evaluate(tSet);
    // Για κάθε επανάληψη του αλγορίθμου
    for (int i = 0; i < maxRounds; i++) {
      // Μεταβολή των θέσεων των μορίων στον χώρο αναζήτησης
      pop.crossOver();
      pop.updateWeights();
      if (i % 10 == 0) {
        output.append(".");
      }
      // Αξιολόγηση του νέου πληθυσμού
      pop.evaluate(tSet);
    }

  }

  private int findBestAtomPos() {

    int pos = -1;
    double maxEval = 0;
    for (int i = 0; i < pop.getPopSize(); i++) {
       if (pop.getAtom(i) != null && pop.getAtom(i).getEval() > maxEval) {
         maxEval = pop.getAtom(i).getEval();
         pos = i;
       }
    }
    return pos;

  }

  public Particle findBestAtom() {
```

```
        return (Particle) pop.getAtom(this.findBestAtomPos());

    }

    public void setOutput(JTextArea output) {
        this.output = output;
    }

    public int getMaxRounds() {
        return maxRounds;
    }

    public double getC1() {
        return c1;
    }

    public double getC2() {
        return c2;
    }

}
```

```
package hgapso;

import neuralnetwork.NeuralNetwork;
import neuralnetwork.TrainingSet;

public class PSOPopulation extends Population {

   // Η καλύτερη θέση στην οποία έχει βρεθεί κάποιο μόριο του σμήνους
   private double[] globalBest;
   private double evalGlobalBest = -1;
   // Παράμετροι c1 και c2 του τύπου ανανέωσης της θέσης ενός μορίου
   private double c1;
   private double c2;

   private String networkXMLFile;
   private TrainingSet tSet;
   private int hiddenLayersNo;
   private int nodesPerLayer;

   // Constructors
   public PSOPopulation(int popSize, double c1, double c2, String networkXMLFile) {
      this.popSize = popSize;
      this.c1 = c1;
      this.c2 = c2;
      this.networkXMLFile = networkXMLFile;
   }

   public PSOPopulation(int popSize, double c1, double c2, TrainingSet tSet,
                  int hiddenLayersNo, int nodesPerLayer) {
      this.popSize = popSize;
      this.c1 = c1;
      this.c2 = c2;
      this.tSet = tSet;
      this.hiddenLayersNo = hiddenLayersNo;
      this.nodesPerLayer = nodesPerLayer;
   }

   @Override
   public void initializePopulation() {
      atoms = new Genotype[popSize];
      for (int i = 0; i < atoms.length; i++) {
         NeuralNetwork network = new NeuralNetwork();
         if (networkXMLFile != null)
            network.readXMLFile(networkXMLFile);
```

```
            else
                network.createNetworkByTrainingSet(tSet,hiddenLayersNo,nodesPerLayer);
            network.randomInitWeights(PSOAlgorithm.MIN_INIT_VAL,PSOAlgorithm.MAX_INIT_VAL);
            atoms[i] = new Particle("pso" + (i+1));
            atoms[i].setNeuralNetwork(network);
            atoms[i].init();
        }
    }

    public void initializePopulation(GAPopulation gapop) {
        atoms = new Genotype[popSize];
        for (int i = 0; i < atoms.length; i++) {
            NeuralNetwork network = gapop.getAtom(i).getNeuralNetwork();
            atoms[i] = new Particle("pso" + (i+1));
            atoms[i].setNeuralNetwork(network);
            atoms[i].init();
        }
    }

    // Αξιολόγηση όλων των μορίων του σμήνους
    public void evaluate(TrainingSet tSet) {
        // Στην περίπτωση που δεν έχει βρεθεί ακόμα το globalBest, θέσε το ίσο
        // με την θέση του πρώτου μορίου του πληθυσμού
        if (globalBest == null)
            globalBest = atoms[0].getGenes();
        // Για κάθε μόριο του πληθυσμού
        for (Genotype g : atoms) {
            Particle p = (Particle) g;
            // Έλεγξε αν θα πρέπει να ανανεωθεί η θέση personalBest του μορίου
            p.evaluate(tSet);
            if (p.getEval() > p.getEvalPersonalBest()) {
                p.setPersonalBest(p.getGenes());
                p.setEvalPersonalBest(p.getEval());
            }
            // Έλεγξε αν θα πρέπει να ανανεωθεί η θέση globalBest του σμήνους
            if (p.getEval() > evalGlobalBest) {
                globalBest = new double[p.getGenes().length];
                for (int i = 0; i < globalBest.length; i++)
                    globalBest[i] = p.getGenes()[i];
                evalGlobalBest = p.getEval();
            }
        }

        // Ενημέρωσε όλα τα μόρια του σμήνους για τυχόν νέα θέση globalBest
```

```
        for (Genotype g : atoms) {
            Particle p = (Particle) g;
            p.setGlobalBest(globalBest);
        }
    }

    // Ανανέωση των θέσεων όλων των μορίων του σμήνους
    public void crossOver() {

        for (Genotype g : atoms) {
            Particle p = (Particle) g;
            p.nextPosition(c1,c2);
        }

    }

    // Getter
    public double[] getGlobalBest() {
        return globalBest;
    }

    @Override
    public void updateWeights() {

        for (Genotype a : atoms)
            ((Particle) a).updateWeights();

    }

}
```

```
package hgapso;

import java.util.Random;
import neuralnetwork.NeuralNetwork;

public class Particle extends Genotype {

   // Η καλύτερη θέση στην οποία έχει βρεθεί το μόριο
   private double[] personalBest;
   // Η καλύτερη θέση του πληθυσμού
   private double[] globalBest;
   // Η ταχύτητα του μορίου
   private double[] v;
   // Η αξιολόγηση του personalBest
   private double evalPersonalBest = -1;

   public Particle(String id) {
      super(id);
   }

   public Particle(String id, NeuralNetwork neuralNetwork) {
      super(id, neuralNetwork);
   }

   public double[] getPersonalBest() {
      return personalBest;
   }

   public void setPersonalBest(double[] personalBest) {
      this.personalBest = new double[personalBest.length];
      for (int i = 0; i < this.personalBest.length; i++)
          this.personalBest[i] = personalBest[i];
   }

   public void setGlobalBest(double[] globalBest) {
      this.globalBest = new double[globalBest.length];
      for (int i = 0; i < this.globalBest.length; i++)
          this.globalBest[i] = globalBest[i];
   }

   public double getEvalPersonalBest() {
      return evalPersonalBest;
   }
```

```
    public void setEvalPersonalBest(double evalPersonalBest) {
        this.evalPersonalBest = evalPersonalBest;
    }

    @Override
    public void init() {

        super.init();
        v = new double[genes.length];
        this.setPersonalBest(genes);

    }

    // Υπολογίζει την επόμενη θέση ενός μορίου. Η μέθοδος δέχεται σαν όρισμα
    // τις παραμέτρους c1 και c2 του τύπου υπολογισμού
    public void nextPosition(double c1, double c2) {

        // Ανάκτηση των διανυσμάτων της τρέχουσας θέσης, της καλύτερης θέσης
        // στην οποία το μόριο έχει βρεθεί και της καλύτερης θέσης όλου
        // του σμήνους
        double[] x = genes;
        double[] pb = personalBest;
        double[] gb = globalBest;

        // Παραγωγή των δύο τυχαίων αριθμών r1 και r2
        Random r = new Random();
        double r1 = r.nextDouble();
        double r2 = r.nextDouble();

        // v(n+1) = v(n) + c1*r1*(pb-x) + c2*r2*(gb-x)
        for (int i = 0; i < x.length; i++) {
            v[i] = v[i] + c1*r1*(pb[i]-x[i]) + c2*r2*(gb[i]-x[i]);
        }

        // x(n+1) = x(n) + v(n+1)
        for (int i = 0; i < x.length; i++) {
            x[i] = x[i] + v[i];
        }

    }

}
```

```
package hgapso;

public abstract class Population {

   protected int popSize;
   protected Genotype[] atoms;

   public Genotype getAtom(int pos) {
      if (pos >= 0 && pos < popSize)
         return atoms[pos];
      else
         return null;
   }

   public void setAtom(int pos, Genotype atom) {
      if (pos >= 0 && pos < popSize)
         atoms[pos] =  atom;
   }

   public int getPopSize() {
      return popSize;
   }

   public abstract void updateWeights();

   public void printPopulation() {

      for (Genotype a : atoms)
          System.out.println(a);

   }

   public abstract void initializePopulation();

}
```

```
package hgapso;

import java.text.DecimalFormat;

public class Utils {

   @SuppressWarnings("empty-statement")
   public static double randValue() {
      double val;
      while ((val = Math.random()) == 0.0);
      return val;
   }

   public static double dblRound2(double d) {
      DecimalFormat twoDForm = new DecimalFormat("#.##");
      return Double.valueOf(twoDForm.format(d).replace(',', '.'));
   }

}
```

```
package neuralnetwork;

public class Connection {

   private Neuron from;
   private Neuron to;
   private double weight;

   public Connection(Neuron from, Neuron to, double weight) {
      this.from = from;
      this.to = to;
      this.weight = weight;
   }

   public Connection(Neuron from, Neuron to) {
      this.from = from;
      this.to = to;
   }

   public Neuron getFrom() {
      return from;
   }

   public Neuron getTo() {
      return to;
   }

   public double getWeight() {
      return weight;
   }

   public void setFrom(Neuron from) {
      this.from = from;
   }

   public void setTo(Neuron to) {
      this.to = to;
   }

   public void setWeight(double weight) {
      this.weight = weight;
   }

}
```

```
package neuralnetwork;

public class Functions {

   public static double sigmoid(double x) {
      return 1.0 / (1.0 + Math.pow(Math.E, -x));
   }

   public static double sigmoidDerivative(double x) {
      return sigmoid(x) * (1 - sigmoid(x));
   }

}
```

```
package neuralnetwork;

public class InputNeuron extends Neuron {

   private double input;

   public InputNeuron(String id) {
      super(id);
   }

   public double getInput() {
      return input;
   }

   public void setInput(double input) {
      this.input = input;
   }

   @Override
   public double output() {
      return input;
   }

}
```

```
package neuralnetwork;

import java.io.File;
import java.util.ArrayList;
import java.util.Random;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class NeuralNetwork {

   private double learningRate;
   public ArrayList<Neuron> neurons;
   public ArrayList<InputNeuron> inputNeurons;
   public ArrayList<Neuron> outputNeurons;

   public NeuralNetwork() {
      neurons = new ArrayList<>();
      inputNeurons = new ArrayList<>();
      outputNeurons = new ArrayList<>();
   }

   public double getLearningRate() {
      return learningRate;
   }

   public void setLearningRate(double learningRate) {
      this.learningRate = learningRate;
   }

   public void addNeuron(Neuron n) {
      neurons.add(n);
   }

   public ArrayList<Neuron> getNeurons() {
      return neurons;
   }

   public void randomInitWeights(double min, double max) {

      Random r = new Random();
```

```
    for (Neuron n : neurons) {
        for (Connection conn : n.getOutgoingConnections()) {
            double randomWeight = min + r.nextInt((int)(max-min)) + r.nextDouble();
            conn.setWeight(randomWeight);
        }
    }

}

public Neuron findNeuronById(String id) {
    for (Neuron n : neurons) {
        if (n.getId().equals(id))
            return n;
    }
    return null;
}

public void findInputNeurons() {
    for (Neuron n : neurons) {
        if (n instanceof InputNeuron)
            inputNeurons.add((InputNeuron) n);
    }
}

public void findOutputNeurons() {
    outputNeurons.clear();
    for (Neuron n : neurons) {
        if (n.isOutputNeuron())
            outputNeurons.add(n);
    }
}

public double[] networkOutput() {
    if (outputNeurons.isEmpty())
       return null;
    double[] output = new double[outputNeurons.size()];
    for (int i = 0; i < outputNeurons.size(); i++)
        output[i] = outputNeurons.get(i).output();
    return output;
}

public void readXMLFile(String xmlFile) {
```

```
try {
  // Άνοιγμα του αρχείου εισόδου
  File inputFile = new File(xmlFile);
  // Δημιουργία του Document
  DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
  DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
  Document doc = dBuilder.parse(inputFile);
  doc.getDocumentElement().normalize();
  // Διαγραφή των προηγούμενων δεδομένων
  neurons.clear();
  inputNeurons.clear();
  outputNeurons.clear();
  // Διάβασμα των δεδομένων
  // Δημιουργία των κόμβων
  NodeList nList = doc.getElementsByTagName("neuron");
  for (int i = 0; i < nList.getLength(); i++) {
    Node nNode = nList.item(i);
    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
      Element eElement = (Element) nNode;
      String id = eElement.getAttribute("id");
      String type = eElement.getElementsByTagName("type").item(0).getTextContent();
      if (type.equals("input")) {
        InputNeuron inputNeuron = new InputNeuron(id);
        neurons.add(inputNeuron);
      } else if (type.equals("calc")) {
        NodeList thresholdList = eElement.getElementsByTagName("threshold");
        double threshold;
        if (thresholdList.getLength() == 0)
          threshold = 0;
        else
          threshold = Double.parseDouble(thresholdList.item(0).getTextContent());
        Neuron neuron = new Neuron(id);
        neuron.setThreshold(threshold);
        neurons.add(neuron);
      }
    }
  }
  // Δημιουργία των συνδέσεων
  for (int i = 0; i < nList.getLength(); i++) {
    Node nNode = nList.item(i);
    if (nNode.getNodeType() == Node.ELEMENT_NODE) {
      Element eElement = (Element) nNode;
      String id = eElement.getAttribute("id");
      Neuron from = this.findNeuronById(id);
```

```
            NodeList connectionList = eElement.getElementsByTagName("connection");
            //System.out.println(id + ": " + connectionList.getLength() + " output connection(s)");
            for (int j = 0; j < connectionList.getLength(); j++) {
                Node connectionNode = connectionList.item(j);
                if (connectionNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element connectionElement = (Element) connectionNode;
                    String outputId =
connectionElement.getElementsByTagName("output").item(0).getTextContent();
                    NodeList weightList = connectionElement.getElementsByTagName("weight");
                    double weight;
                    if (weightList.getLength() == 0)
                      weight = 0;
                    else
                      weight = Double.parseDouble(weightList.item(0).getTextContent());
                    //System.out.println("\t-> " + outputId + ": " + weight);
                    Neuron to = this.findNeuronById(outputId);
                    Connection conn = new Connection(from,to,weight);
                    from.addOutgoingConnection(conn);
                    to.addIncomingConnection(conn);
                }
            }
          }
        }
        // Εύρεση των νευρώνων εισόδου και των νευρώνων εξόδου
        this.findInputNeurons();
        this.findOutputNeurons();
//        System.out.println("Input neurons: ");
//        for (Neuron n : inputNeurons) {
//            System.out.println(n.getId());
//        }
//        System.out.println("Output neurons: ");
//        for (Neuron n : outputNeurons) {
//            System.out.println(n.getId());
//        }
      }
      catch(Exception e) {
          System.out.println(e);
      }

  }

  public void createNetworkByTrainingSet(TrainingSet tSet, int hiddenLayersNo, int nodesPerLayer) {

      int inputsNo = tSet.getInputDimension();
```

```
int outputsNo = tSet.getOutputDimension();

neurons.clear();
inputNeurons.clear();
outputNeurons.clear();

// Δημιουργία των νευρώνων του επιπέδου εισόδου
for (int i = 0; i < inputsNo; i++) {
   InputNeuron inputNeuron = new InputNeuron("Input" + (i+1));
   neurons.add(inputNeuron);
   inputNeurons.add(inputNeuron);
}

// Δημιουργία των νευρώνων των κρυφών επιπέδων
Neuron[] previousLayerNeurons = new Neuron[nodesPerLayer];
for (int i = 0; i < hiddenLayersNo; i++) {
   Neuron[] tmp = new Neuron[nodesPerLayer];
   for (int j = 0; j < nodesPerLayer; j++) {
      Neuron hiddenNeuron = new Neuron("calc" + (i+1) + (j+1));
      neurons.add(hiddenNeuron);
      if (i == 0) {
         for (InputNeuron inputNeuron : inputNeurons) {
            Connection conn = new Connection(inputNeuron,hiddenNeuron);
            inputNeuron.addOutgoingConnection(conn);
            hiddenNeuron.addIncomingConnection(conn);
         }
         previousLayerNeurons[j] = hiddenNeuron;
      } else {
         for (int k = 0; k < nodesPerLayer; k++) {
            Connection conn = new Connection(previousLayerNeurons[k],hiddenNeuron);
            previousLayerNeurons[k].addOutgoingConnection(conn);
            hiddenNeuron.addIncomingConnection(conn);
         }
         tmp[j] = hiddenNeuron;
      }
   }
   if (i > 0)
      previousLayerNeurons = tmp;
}

// Δημιουργία των νευρώνων του επιπέδου εξόδου
for (int i = 0; i < outputsNo; i++) {
   Neuron outputNeuron = new Neuron("output" + (i+1));
   neurons.add(outputNeuron);
```

```
            outputNeurons.add(outputNeuron);
            for (int j = 0; j < nodesPerLayer; j++) {
                Connection conn = new Connection(previousLayerNeurons[j],outputNeuron);
                previousLayerNeurons[j].addOutgoingConnection(conn);
                outputNeuron.addIncomingConnection(conn);
            }
        }

    }

    public double[] forwardPass(double[] x) {

        // Θέσε τις εισόδους στους νευρώνες εισόδου
        for (int i = 0; i < x.length; i++)
            inputNeurons.get(i).setInput(x[i]);
        // Υπολόγισε και επέστρεψε την έξοδο του δικτύου
        return this.networkOutput();

    }

    public void backwardPass(double[] y, double[] d) {

        // Υπολόγισε το σφάλμα για κάθε νευρώνα εξόδου
        double[] e = new double[y.length];
        for (int i = 0; i < e.length; i++)
            e[i] = d[i] - y[i];

        // Υπολογισμός του δέλτα για τον κάθε νευρώνα εξόδου
        for (int i = 0; i < outputNeurons.size(); i++) {
            double delta = e[i] * (y[i] * (1 - y[i]));
            outputNeurons.get(i).setDelta(delta);
        }

        // Υπολογισμός του δέλτα για τον κάθε κρυφό νευρώνα
        for (int i = neurons.size()-1; i >= 0; i--) {
            Neuron n = neurons.get(i);
            if (n.isOutputNeuron() || n instanceof InputNeuron)
                continue;
            double delta = 0.0;
            for (Connection conn : n.getOutgoingConnections()) {
                delta += conn.getWeight() * conn.getTo().getDelta();
            }
            delta *= n.output() * (1 - n.output());
            n.setDelta(delta);
```

```
        }

        // Ανανέωση των βαρών για κάθε υπολογιστικό νευρώνα
        for (Neuron n : neurons) {
            if (n instanceof InputNeuron)
                continue;
            // Ανανέωση των βαρών με μεταβλητές εισόδους
            for (Connection conn : n.getIncomingConnections()) {
                double newWeight = conn.getWeight() + learningRate * n.getDelta() *
conn.getFrom().output();
                conn.setWeight(newWeight);
            }
            // Ανανέωση του κατωφλίου
            double newThreshold = n.getThreshold() + learningRate * n.getDelta() * (-1);
            n.setThreshold(newThreshold);
        }

    }

    public void printWeights() {

        System.out.println("Weights:");
        System.out.println("--------");
        for (Neuron n : neurons) {
            if (n instanceof InputNeuron)
                continue;
            System.out.println("θ" + n.getId() + ":" + n.getThreshold());
            for (Connection conn : n.getIncomingConnections()) {
                System.out.println(conn.getFrom().getId() + "->" + conn.getTo().getId() + ": " +
                        conn.getWeight());
            }
        }

    }

    public int numberOfWeights() {

        int weightsNum = 0;
        for (Neuron n : neurons) {
            weightsNum += n.getOutgoingConnections().size();
            if (!(n instanceof InputNeuron))
                weightsNum++;
        }
        return weightsNum;
```

```
    }

    public double squareError(double[][] inputs, double[][] d) {

        double squareError = 0;
        for (int i = 0; i < inputs.length; i++) {
            double[] output = this.forwardPass(inputs[i]);
            for (int j = 0; j < output.length; j++) {
                squareError += Math.pow(d[i][j]-output[j],2);
            }
        }

        return squareError;

    }

    public double meanSquareError(double[][] inputs, double[][] d, double a) {

        double squareError = 0;
        int n = 0;
        for (int i = 0; i < inputs.length; i++) {
            double[] output = this.forwardPass(inputs[i]);
            for (int j = 0; j < output.length; j++) {
                squareError += Math.pow(d[i][j]-output[j],2);
            }
            n++;
        }

        return (n != 0) ? (squareError / n) * a : squareError * a;

    }

}
```

```
package neuralnetwork;

import java.util.ArrayList;

public class Neuron {

   private String id;
   private double threshold;
   private ArrayList<Connection> incomingConnections;
   private ArrayList<Connection> outgoingConnections;
   private double delta;

   public Neuron(String id) {
      this.id = id;
      incomingConnections = new ArrayList<>();
      outgoingConnections = new ArrayList<>();
   }

   public void addIncomingConnection(Connection c) {
      incomingConnections.add(c);
   }

   public void addOutgoingConnection(Connection c) {
      outgoingConnections.add(c);
   }

   public String getId() {
      return id;
   }

   public void setId(String id) {
      this.id = id;
   }

   public ArrayList<Connection> getIncomingConnections() {
      return incomingConnections;
   }

   public ArrayList<Connection> getOutgoingConnections() {
      return outgoingConnections;
   }

   public double getThreshold() {
      return threshold;
```

```
    }

    public void setThreshold(double threshold) {
        this.threshold = threshold;
    }

    public double getDelta() {
        return delta;
    }

    public void setDelta(double delta) {
        this.delta = delta;
    }

    public double output() {

        double u = 0;
        for (Connection c : incomingConnections) {
            u += c.getWeight() * c.getFrom().output();
        }
        u += -1*threshold;
        return Functions.sigmoid(u);

    }

    public boolean isOutputNeuron() {
        return outgoingConnections.isEmpty();
    }

}
```

```
package neuralnetwork;

import java.util.ArrayList;

public class TrainingExample {

   private ArrayList<Double> input;
   private ArrayList<Double> output;

   public TrainingExample(ArrayList<Double> input, ArrayList<Double> output) {
      this.input = input;
      this.output = output;
   }

   public ArrayList<Double> getInput() {
      return input;
   }

   public void setInput(ArrayList<Double> input) {
      this.input = input;
   }

   public ArrayList<Double> getOutput() {
      return output;
   }

   public void setOutput(ArrayList<Double> output) {
      this.output = output;
   }

}
```

```
package neuralnetwork;

import java.io.File;
import java.util.ArrayList;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

public class TrainingSet {

   private ArrayList<TrainingExample> examples;

   public TrainingSet() {
      examples = new ArrayList<>();
   }

   public void addTrainingExample(TrainingExample te) {
      examples.add(te);
   }

   public int getInputDimension() {

      return this.getInputs()[0].length;

   }

   public int getOutputDimension() {

      return this.getOutputs()[0].length;

   }

   public double[][] getInputs() {

      double[][] inputs = new double[examples.size()][];
      for (int i = 0; i < inputs.length; i++) {
          inputs[i] = new double[examples.get(i).getInput().size()];
          for (int j = 0; j < examples.get(i).getInput().size(); j++)
              inputs[i][j] = examples.get(i).getInput().get(j);
      }
```

```
        return inputs;

    }

    public double[][] getOutputs() {

        double[][] outputs = new double[examples.size()][];
        for (int i = 0; i < outputs.length; i++) {
            outputs[i] = new double[examples.get(i).getOutput().size()];
            for (int j = 0; j < examples.get(i).getOutput().size(); j++) {
                outputs[i][j] = examples.get(i).getOutput().get(j);
            }
        }

        return outputs;

    }

    public void readXMLFile(String xmlFile) {

        try {
            // Άνοιγμα του αρχείου εισόδου
            File inputFile = new File(xmlFile);
            // Δημιουργία του Document
            DocumentBuilderFactory dbFactory = DocumentBuilderFactory.newInstance();
            DocumentBuilder dBuilder = dbFactory.newDocumentBuilder();
            Document doc = dBuilder.parse(inputFile);
            doc.getDocumentElement().normalize();
            // Διάβασμα των δεδομένων
            NodeList exampleList = doc.getElementsByTagName("example");
            // Για κάθε παράδειγμα εκπαίδευσης
            for (int i = 0; i < exampleList.getLength(); i++) {
                Node exampleNode = exampleList.item(i);
                if (exampleNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element exampleElement = (Element) exampleNode;
                    ArrayList<Double> inputs = new ArrayList<>();
                    ArrayList<Double> outputs = new ArrayList<>();
                    // Δεδομένα εισόδου του παραδείγματος εκπαίδευσης
                    NodeList inputList = exampleElement.getElementsByTagName("input");
                    for (int j = 0; j < inputList.getLength(); j++) {
                        Node inputNode = inputList.item(j);
                        if (inputNode.getNodeType() == Node.ELEMENT_NODE) {
                            Element inputElement = (Element) inputNode;
                            Double inputValue = Double.parseDouble(inputElement.getTextContent());
```

```
                    inputs.add(inputValue);
                }
            }
            // Δεδομένα εξόδου του παραδείγματος εκπαίδευσης
            NodeList outputList = exampleElement.getElementsByTagName("output");
            for (int j = 0; j < outputList.getLength(); j++) {
                Node outputNode = outputList.item(j);
                if (outputNode.getNodeType() == Node.ELEMENT_NODE) {
                    Element outputElement = (Element) outputNode;
                    Double outputValue = Double.parseDouble(outputElement.getTextContent());
                    outputs.add(outputValue);
                }
            }
            // Δημιουργία του παραδείγματος εκπαίδευσης και
            // προσθήκη του στο σύνολο με τα παραδείγματα εκπαίδευσης
            TrainingExample example = new TrainingExample(inputs,outputs);
            examples.add(example);
          }
        }
    }
    catch(Exception e) {
       System.out.println(e);
    }

}

  public void printSet() {

    double[][] inputs = this.getInputs();
    double[][] outputs = this.getOutputs();

    System.out.println("Inputs: ");
    for (int i = 0; i < inputs.length; i++) {
        System.out.print("[ ");
        for (int j = 0; j < inputs[i].length; j++) {
            System.out.print(inputs[i][j] + " ");
        }
        System.out.println("]");
    }

    System.out.println("Outputs: ");
    for (int i = 0; i < outputs.length; i++) {
        System.out.print("[ ");
        for (int j = 0; j < outputs[i].length; j++) {
```

```
            System.out.print(outputs[i][j] + " ");
        }
        System.out.println("]");
    }

}

public TrainingSet trainSet(double p) {

    TrainingSet tSet = new TrainingSet();

    int tSetSize = (int) (examples.size() * p);

    for (int i = 0; i < tSetSize; i++) {
        ArrayList<Double> input = new ArrayList(examples.get(i).getInput());
        ArrayList<Double> output = new ArrayList(examples.get(i).getOutput());
        TrainingExample te = new TrainingExample(input,output);
        tSet.addTrainingExample(te);
    }

    return tSet;

}

public TrainingSet testSet(double p) {

    TrainingSet tSet = new TrainingSet();

    int tSetSize = (int) (examples.size() * p);
    int startIndex = examples.size() - tSetSize;

    for (int i = startIndex; i < examples.size(); i++) {
        ArrayList<Double> input = new ArrayList(examples.get(i).getInput());
        ArrayList<Double> output = new ArrayList(examples.get(i).getOutput());
        TrainingExample te = new TrainingExample(input,output);
        tSet.addTrainingExample(te);
    }

    return tSet;

}

}
```

```
package trainingalgorithms;

import javax.swing.JTextArea;
import neuralnetwork.NeuralNetwork;
import neuralnetwork.TrainingSet;

public class BackPropagationAlgorithm extends TrainingAlgorithm {

   private static final double DEFAULT_LEARNING_RATE = 0.5;
   private static final int DEFAULT_MAX_EPOCHS = 10000;

   private double learningRate;
   private int maxEpochs;

   private JTextArea output;

   public BackPropagationAlgorithm(NeuralNetwork network) {
      super(network);
      maxEpochs = DEFAULT_MAX_EPOCHS;
   }

   public BackPropagationAlgorithm(NeuralNetwork network, int maxEpochs) {
      super(network);
      learningRate = DEFAULT_LEARNING_RATE;
      this.maxEpochs = maxEpochs;
   }

   public double getLearningRate() {
      return learningRate;
   }

   public int getMaxEpochs() {
      return maxEpochs;
   }

   public void setMaxEpochs(int maxEpochs) {
      this.maxEpochs = maxEpochs;
   }

   public void setLearningRate(double learningRate) {
      this.learningRate = learningRate;
   }

   public void setOutput(JTextArea output) {
```

```
        this.output = output;
    }

    @Override
    public void execute(TrainingSet tSet) {

        double[][] inputs = tSet.getInputs();
        double[][] d = tSet.getOutputs();

        network.setLearningRate(learningRate);

        // Για κάθε εποχή εκπαίδευσης
        for (int epoch = 1; epoch <= maxEpochs; epoch++) {
            if (epoch % 100 == 0)
               output.append(".");
            // Για κάθε στιγμιότυπο εκπαίδευσης
            for (int i = 0; i < inputs.length; i++) {
                // Πέρασμα προς τα εμπρός
                double[] y = network.forwardPass(inputs[i]);
                // Πέρασμα προς τα πίσω
                network.backwardPass(y,d[i]);
            }
        }


    }

}
```

```
package trainingalgorithms;

import neuralnetwork.NeuralNetwork;
import neuralnetwork.TrainingSet;

public abstract class TrainingAlgorithm {

   protected NeuralNetwork network;

   public TrainingAlgorithm(NeuralNetwork network) {
      this.network = network;
   }

   public abstract void execute(TrainingSet tSet);

}
```