Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Εξόρυξη δικτυακών εισβολών με χρήση Γενετικών Αλγορίθμων**
Ονοματεπώνυμο Φοιτητή	**Γεώργιος Χρυσολωράς**
Πατρώνυμο	**Θεόδωρος**
Αριθμός Μητρώου	**ΜΠΣΠ/08019**
Επιβλέπων	**Χρήστος Δουληγέρης, Καθηγητής**

Ημερομηνία Παράδοσης **Οκτώβριος 2011**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Χρήστος Δουληγέρης Καθηγητής	Δέσποινα Πολέμη Επίκουρη Καθηγήτρια	Παναγιώτης Κοτζανικολάου Λέκτορας

Περίληψη

Τα δίκτυα υπολογιστών έχουν να αντιμετωπίσουν σε καθημερινή βάση απειλές ασφάλειας και επιθέσεις από εξωτερικούς και εσωτερικούς παράγοντες. Παρόλη την τεχνολογική εξέλιξη στον τομέα της ασφάλειας δικτύων, παραμένει δύσκολη διαδικασία η προστασία των δικτύων των υπολογιστικών συστημάτων. Τα συστήματα ανίχνευσης εισβολών αποτελούν ένα σημαντικό εργαλείο στην ανίχνευση και αντιμετώπιση κακόβουλων επιθέσεων στα υπολογιστικά συστήματα ενός δικτύου. Διάφορες ευφυείς τεχνικές (soft computing) έχουν προταθεί για την ανίχνευση δικτυακών εισβολών, υπάρχει όμως περιθώριο βελτίωσης και έρευνας όσον αφορά την ταχύτητα, ακρίβεια και προσαρμοστικότητα των τεχνικών ανίχνευσης.

Στην εργασία αυτή παρουσιάζεται μια προσέγγιση ενός Γενετικού Αλγόριθμου στην ανίχνευση εισβολών και η υλοποίησή του σε λογισμικό. Ο Γενετικός αλγόριθμος εκπαιδεύεται σε ορισμένα δεδομένα καταγραφής του δικτύου ώστε να παράγει ένα σύνολο κανόνων εισβολής. Οι κανόνες που εξορύσσονται, εφαρμόζονται για να ταξινομήσουν την κίνηση ενός υποτιθέμενου πραγματικού περιβάλλοντος σε “φυσιολογική κίνηση” ή “εισβολή”. Για την εκπαίδευση και ανίχνευση χρησιμοποιείται η συλλογή δεδομένων KDD99Cup. Το πλήθος των δικτυακών χαρακτηριστικών που χρησιμοποιείται για την εκπαίδευση και ανίχνευση εισβολών, είναι μικρό ώστε η ανίχνευση να γίνεται γρήγορα και η εκπαίδευση του αλγορίθμου για εξαγωγή νέων κανόνων να είναι εφικτή σε πραγματικό περιβάλλον. Τα αποτελέσματα της προσέγγισης, βασιζόμενα στην συλλογή δεδομένων KDD99, είναι αποδεκτά, διατηρώντας σε υψηλά επίπεδα την ανίχνευση επιθέσεων, ενώ παράλληλα ο χρόνος εκπαίδευσης είναι μικρός.

Λέξεις κλειδιά: ανίχνευση εισβολών, γενετικοί αλγόριθμοι, εξόρυξη δεδομένων, ανάπτυξη λογισμικού.

Abstract

Computer networks have to cope with security threats and attacks on an everyday basis. Despite the contemporary tools and methods built for security assurance, it is still difficult to protect computer networks. Intrusion detection systems play an important role in identifying malevolent actions and attacks, while acting as a tool for a better security policy. There are various soft computing approaches in detecting intrusions to a computer network. Still, there are a lot of possibilities for these techniques to be improved in terms of speed, accuracy and adaptability.

In this work a Genetic Algorithm approach is presented that detects intrusions and its software implementation. The Genetic Algorithm is trained towards some network audit data in order to derive a set of intrusion rules. The mined rules are applied to classify network activity into “normal” or “intrusion”. The KDD99Cup training dataset is used for training the algorithm. The number of network features used was small in order to speed up the training and detection procedure and to be feasible in a real world environment. The results using KDD99 dataset are acceptable, keeping the detection rate high while maintaining small training time.

keywords: intrusion detection, genetic algorithms, data mining, software development.

Πίνακας Περιεχομένων

Κατάλογος Πινάκων

Κατάλογος Σχημάτων

1. Εισαγωγή – Σύντομη περιγραφή

Τα δίκτυα υπολογιστών, είτε με τη μορφή του διαδικτύου, είτε με τη μορφή διασύνδεσης στα πλαίσια ενός οργανισμού, αντιμετωπίζουν σε καθημερινή βάση πληθώρα απειλών, πολλές εκ των οποίων είναι παραλλαγές υπαρχόντων ή καινούριες. Στα εργαλεία του διαχειριστή ασφάλειας υπάρχει πληθώρα εργαλείων για την ενίσχυση της πολιτικής ασφάλειας όπως λογισμικό προστασίας από ιούς, αναχώματα ασφάλειας, κρυπτογράφηση, λίστες ελέγχου πρόσβασης, ασφαλή πρωτόκολλα επικοινωνίας κ.α. Παρόλα αυτά η πλήρης ασφάλεια είναι δύσκολο έως αδύνατο να επιτευχθεί καθώς ένας επίδοξος επιτιθέμενος μπορεί πάντα να βρει τρόπο να διεισδύσει σε ένα δίκτυο. Ένα εργαλείο που προσθέτει ένα πρόσθετο επίπεδο στην ασφάλεια ενός δικτύου είναι τα συστήματα ανίχνευσης εισβολών (intrusion detection systems – IDS). Τα συστήματα ανίχνευσης εισβολών παρακολουθούν τη δραστηριότητα του δικτύου και αναγνωρίζουν παραβιάσεις όπως περίεργη συμπεριφορά, μη εξουσιοδοτημένη πρόσβαση, προσπάθεια εισβολής σε σύστημα του δικτύου, βολιδοσκόπηση υπηρεσιών του δικτύου ακόμα και ανάρμοστη συμπεριφορά από το εσωτερικό του δικτύου.

Τα υπάρχοντα συστήματα ανίχνευσης εισβολών προσφέρουν πλήθος παραμετροποιήσεων με βάση την εκάστοτε πολιτική ασφάλειας ενός δικτύου, έχουν αρκετά υψηλή απόδοση όσον αφορά την ανίχνευση εισβολών – ειδικά των υπαρχόντων και καλά γνωστών. Παρόλα αυτά, υπάρχει συνεχώς ανάγκη για βελτίωση σε διάφορα θέματα. Ορισμένα βασικά είναι η ταχύτητα, η ακρίβεια, και η προσαρμοστικότητα. Η ταχύτητα αφορά το πολύ μεγάλο πλήθος δεδομένων που χρειάζονται για την παρακολούθηση του δικτύου ώστε να αποφανθούν για επιθέσεις. Η ακρίβεια αφορά την αύξηση του ποσοστού ανίχνευσης εισβολών και ταυτόχρονα την μείωση της όχλησης από λανθασμένες θετικές προειδοποιήσεις (false positives), δηλαδή φυσιολογική συμπεριφορά που εκλαμβάνεται ως εισβολή. Η προσαρμοστικότητα αφορά την δυνατότητα ανίχνευσης πρωτοεμφανιζόμενων εισβολών ή εισβολές διαφοροποιημένες όσον αφορά τα δικτυακά χαρακτηριστικά τους.

Στην εργασία αυτή, παρουσιάζεται η σχεδίαση και υλοποίηση μιας προσέγγισης ενός Γενετικού Αλγόριθμου στην ταξινόμηση δικτυακών εισβολών σε ένα δίκτυο. Οι Γενετικοί Αλγόριθμοι έχουν ευελιξία, μεγάλο βαθμό ανεξαρτησίας από το υπό εξέταση πρόβλημα, είναι προσαρμόσιμοι και έχουν καλά αποτελέσματα σε προβλήματα ταξινόμησης που αφορούν βελτιστοποίηση και δυσκολία στην αναζήτηση σε πολύ μεγάλο εύρος πιθανών λύσεων.

Στην προσέγγιση αυτή γίνεται προσπάθεια για διατήρηση υψηλών ποσοστών εξόρυξης εισβολών ενώ, για την αύξηση της ταχύτητας χρησιμοποιείται μικρό υποσύνολο δικτυακών χαρακτηριστικών.

Η προτεινόμενη προσέγγιση υλοποιείται με εκπαίδευση του αλγορίθμου σε ένα μικρό σύνολο δεδομένων ώστε να εξαχθούν κανόνες εισβολής και έπειτα εξέταση των κανόνων εισβολής σε ένα μεγάλο σύνολο για την αξιολόγηση της ανίχνευσης. Για την εκπαίδευση και αξιολόγηση χρησιμοποιήθηκε η συλλογή δεδομένων KDD99Cup [1], η οποία αποτελεί την πιο ευρέως χρησιμοποιούμενη συλλογή δεδομένων για την δοκιμή και αξιολόγηση συστημάτων ανίχνευσης εισβολών. Η μείωση των διαστάσεων των δικτυακών χαρακτηριστικών έγινε με βάση την μέθοδο PCA (Principal Component Analysis) την οποία παρουσίασε η Bancovic [2] για τη συλλογή δεδομένων KDD99Cup. Η εκπαίδευση του αλγορίθμου λόγω της μείωσης των διαστάσεων δικτυακών χαρακτηριστικών και τη χρησιμοποίηση μικρού υποσυνόλου δεδομένων εκπαίδευσης θεωρείται γρήγορη και εφαρμόσιμη σε ένα πραγματικό περιβάλλον.

Η δομή της εργασίας οργανώνεται με τον ακόλουθο τρόπο. Στην ενότητα 2, γίνεται μια συνοπτική επισκόπηση στα συστήματα ανίχνευσης εισβολών, οι λόγοι χρήσης τους, οι στόχοι που πρέπει να έχουν στη σχεδίαση και υλοποίησή τους, η ταξινόμησή τους με βάση τις τεχνικές ανίχνευσης, το γενικότερο μοντέλο και η λειτουργική δομή τους.

Στην ενότητα 0, γίνεται επισκόπηση των Γενετικών Αλγορίθμων, περιγράφονται τα πλεονεκτήματά τους, η δομή τους, οι λόγοι που χρησιμοποιούνται καθώς και διάφορα θέματα που αφορούν την υλοποίησή τους.

Στην ενότητα 0, παρουσιάζεται η προσέγγιση ενός Γενετικού Αλγόριθμου στην εξόρυξη εισβολών. Παρουσιάζεται η σχετική έρευνα των μεθόδων μηχανικής εκμάθησης που προέρχονται από το χώρο της τεχνικής νοημοσύνης και συγκεκριμένα τις Soft Computing τεχνικές, συμπεριλαμβανομένων των Γενετικών Αλγορίθμων, Παρουσιάζεται το πρόβλημα και η σχεδίαση του αλγορίθμου καθώς και οι

βασικές παράμετροι σχεδίασης του, όπως η αναπαράστασή του και η αντικειμενική συνάρτηση που χρησιμοποιείται.

Στην ενότητα 0, περιγράφονται οι προδιαγραφές και τα τεχνικά χαρακτηριστικά με τα οποία υλοποιήθηκε ο Γενετικός Αλγόριθμος. Περισσότερα στοιχεία υλοποίησης παρατίθενται στο παράρτημα Α.

Στην ενότητα 0, καταγράφεται η διαδικασία και οι παράμετροι που χρησιμοποιήθηκαν από το πρόγραμμα, τα αποτελέσματα από τις μετρήσεις και από την αξιολόγηση του αλγορίθμου.

Στην ενότητα 0, εξάγονται συμπεράσματα και συζητείται ενδεχόμενος εμπλουτισμός της παρούσας προσέγγισης.

2. Συστήματα ανίχνευσης δικτυακών εισβολών

Στην ενότητα αυτή, γίνεται μια συνοπτική παρουσίαση των συστημάτων ανίχνευσης εισβολών, από τι προέκυψε η ανάγκη τους στην υποδομή ασφάλειας ενός οργανισμού και τους στόχους που πρέπει να έχουν στη σχεδίαση και υλοποίηση τους. Επιπλέον, γίνεται μια ταξινόμηση των τεχνικών ανίχνευσης και παρουσιάζεται το μοντέλο και η βασική λειτουργική δομή ενός συστήματος ανίχνευσης δικτυακών εισβολών.

Από την δημιουργία της έννοιας της συνδεσιμότητας υπολογιστών, εισήχθη η έννοια της ασφάλειας των δικτύων υπολογιστών. Με τον όρο ασφάλεια [3], ορίζεται:

- Μια κατάσταση η οποία δεν έχει ρίσκο ή αίσθηση απειλής
- Η παρεμπόδιση του ρίσκου μιας απειλής
- Η διασφάλιση μια αίσθησης εμπιστοσύνης και βεβαιότητας

Για την ασφάλεια σε ένα δίκτυο υπολογιστικών συστημάτων η ασφάλεια περιγράφεται από τα παρακάτω χαρακτηριστικά [4]: εμπιστευτικότητα (confidentiality), ακεραιότητα (integrity) και διαθεσιμότητα (availability). Η εμπιστευτικότητα αποτελεί το χαρακτηριστικό το οποίο προστατεύει την πληροφορία που μεταδίδεται από όλους, εκτός από τους νόμιμους δέκτες της. Η ακεραιτότητα προστατεύει την μεταδιδόμενη πληροφορία από αλλοίωση από μη εξουσιοδοτημένους χρήστες. Η διαθεσιμότητα προστατεύει την πληροφορία ή τις δικτυακές υπηρεσίες από προσωρινή ή μόνιμη στέρηση παροχής.

Άλλα βασικά χαρακτηριστικά στην ασφάλεια δικτύων είναι η πιστοποίηση (authentication) και η μη αποποίηση ευθύνης (non repudiation). Η πιστοποίηση, επιβεβαιώνει την ταυτότητα μιας οντότητας ή την πηγή μιας πληροφορίας. Η μη αποποίηση ευθύνης εξασφαλίζει ότι ο κύριος μιας ενέργειας, δεν μπορεί να αρνηθεί μετέπειτα ότι τη διέπραξε.

Η αδυναμία τήρησης των παραπάνω βασικών χαρακτηριστικών ασφάλειας, εκθέτει τα υπολογιστικά συστήματα ενός δικτύου σε απειλές. Για την πραγματοποίηση των απειλών οι επιτιθέμενοι προσπαθούν να εκμεταλλευθούν την αδυναμία που προήλθε από τη μη τήρηση των χαρακτηριστικών ασφάλειας ή από την καταστρατήγηση της πολιτικής ασφάλειας με σκοπό την πρόσβαση σε πόρους, πληροφορίες ή την παρεμπόδιση της σωστής λειτουργίας του δικτύου και των υπολογιστικών συστημάτων του.

2.1. Τι είναι τα συστήματα ανίχνευσης εισβολών

Ανίχνευση εισβολών σε ένα δίκτυο [5], είναι η διαδικασία της παρακολούθησης των γεγονότων που συμβαίνουν στο δίκτυο και η ανάλυσή τους για ίχνη εισβολών, προσπαθειών δηλαδή για καταστρατήγηση των παραπάνω χαρακτηριστικών ασφάλειας που αναφέρθηκαν.

Ως επίθεση [6], χαρακτηρίζουμε μια συγκεκριμένη εκτέλεση ενός σχεδίου με σκοπό να πραγματοποιηθεί μια προσπάθεια απειλής. Ως εργαλείο επίθεσης (attack tool) [7], χαρακτηρίζεται ένα αυτοματοποιημένο πρόγραμμα, το οποίο είναι σχεδιασμένο με σκοπό την παραβίαση της πολιτικής ασφάλειας ενός συστήματος. Μια εισβολή (intrusion) είναι μια επιτυχημένη επίθεση η οποία λαμβάνει χώρα και σκοπό έχει τη μη εξουσιοδοτημένη χρήση ή τη δολιοφθορά πόρων του δικτύου. Άλλη προσέγγιση για την εισβολή, εσωκλείοντας και την έννοια της επίθεσης έχει ως εξής [8]: Εισβολή ορίζεται κάθε σύνολο ενεργειών που προσπαθούν να διαβάλλουν την ακεραιότητα, την εμπιστευτικότητα ή τη διαθεσιμότητα ενός υπολογιστικού πόρου. Συνεπώς κάθε επίθεση μπορεί να θεωρηθεί και ως εισβολή, ανεξάρτητα από την επιτυχία ή αποτυχία της.

Μια επίθεση σε ένα δίκτυο, μπορεί να συμβεί είτε από εξωτερικούς παράγοντες, οι οποίοι δεν έχουν εξουσιοδότηση για τη χρήση των πόρων του δικτύου, είτε από εσωτερικούς παράγοντες, οι οποίοι έχουν δικαιώματα πρόσβασης, παρόλα αυτά επιχειρούν να καταστρατηγήσουν την πολιτική ασφάλειας του δικτύου.

Για την ασφάλεια ενός δικτύου, υπάρχουν διάφορα εργαλεία προστασίας. Τα βασικότερα από αυτά είναι ο έλεγχος πρόσβασης (access control), το λογισμικό προστασίας από Ιούς (antivirus software), και τα αναχώματα προστασίας (firewalls). Έρευνες [9], [10], [11], [12] έχουν δείξει ότι τα συστήματα

ανίχνευσης εισβολών είναι μέσα στις πρώτες επιλογές τεχνολογιών ασφάλειας που χρησιμοποιούν οι οργανισμοί (Πίνακας 1).

	2008	2006	2005	2004
Antivirus	97%	97%	96%	99%
Firewalls	94%	98%	97%	98%
Intrusion Detection Systems	69%	69%	72%	68%

Πίνακας 1: Ποσοστό τεχνολογιών ασφάλειας που χρησιμοποιήθηκαν από οργανισμούς [9], [10], [11], [12]

2.2. Λόγοι χρήσης συστημάτων ανίχνευσης εισβολών

Δύο σημαντικά πλεονεκτήματα της ασφάλειας δικτύων μπορούν να θεωρηθούν η *ορατότητα* και ο *έλεγχος* [13]. Ο έλεγχος παρέχεται μέσα από τα αναχώματα ασφαλείας (firewalls) από λίστες πρόσβασης στους δρομολογητές και άλλους τρόπους. Ο έλεγχος ουσιαστικά είναι η υλοποίηση της πολιτικής ασφάλειας αλλά συχνά λόγο ανθρώπινου παράγοντα, τρύπες ασφάλειας παραμένουν στην πολιτική. Η ορατότητα, που ένα σύστημα ανίχνευσης δικτυακών επιθέσεων μπορεί να προσφέρει, επιτρέπει την αναγνώριση τέτοιων ευπαθειών και δίνει την απαραίτητη πληροφορία για την παραμετροποίηση των συστημάτων ελέγχου αντίστοιχα. Για παράδειγμα [14], ένα σύστημα ανίχνευσης δικτυακών εισβολών είναι τοποθετημένο έξω από το ανάχωμα ασφαλείας ενός δικτύου οργανισμού. Κάποια στιγμή, το σύστημα ενεργοποιεί ένα συναγερμό ότι διάφορες υπηρεσίες βολιδοσκοπούνται και οι οποίες όμως έχουν ήδη αποτραπεί από το ανάχωμα ασφαλείας. Ένας έμπειρος διαχειριστής βολιδοσκοπεί ο ίδιος ώστε να ελέγξει ποιες υπηρεσίες σε ποια υπολογιστικά μηχανήματα υπάρχουν με ευπάθεια και ανακαλύπτει διάφορα από αυτά. Παρόλο που τα υπολογιστικά συστήματα δεν εκτέθηκαν σε κίνδυνο, η ασφάλεια δικτύου του οργανισμού βελτιώθηκε, καθώς το σύστημα ανίχνευσης δικτυακών εισβολών παρείχε ορατότητα σε πιθανή απειλή. Επιπρόσθετα, το ανάχωμα ασφαλείας μπορεί να προστάτευσε την επίθεση από τον έξω κόσμο, όμως δεν θα μπορούσε να εμποδίσει την εκμετάλλευση των ευπαθών υπηρεσιών από το εσωτερικό δίκτυο ή από μετάδοση με διαφορετικούς τρόπους (π.χ. e-mail).

Ένας άλλος σημαντικός λόγος που κάνει την ύπαρξη αναχωμάτων ασφαλείας μη επαρκή από μόνη της [15], είναι και τα προγραμματιστικά λάθη (bugs) των διαφόρων προγραμμάτων που χρησιμοποιεί ένας οργανισμός και τα οποία μπορεί να καταλήγουν σε πρόβλημα ασφάλειας. Τέτοια προγραμματιστικά λάθη μπορεί να υπάρχουν είτε σε ένα λειτουργικό σύστημα, είτε σε έναν εξυπηρετητή διαδικτύου, είτε και στο ίδιο το ανάχωμα ασφάλειας. Χαρακτηριστικό παράδειγμα είναι ένα προγραμματιστικό λάθος του προγράμματος εξυπηρετητή του Microsoft το Internet Information Server (IIS), το οποίο ανακαλύφθηκε το 1997 και μπορούσε κάποιος με μικρή προσπάθεια να στείλει μια πολύ μεγάλη διεύθυνση (URL) και ο εξυπηρετητής να κολλήσει. Αυτό είναι ένα χαρακτηριστικό παράδειγμα από το οποίο η ύπαρξη πρώτου επιπέδου ασφάλειας (π.χ. αναχώματα ασφάλειας) δεν είναι πάντα επαρκής.

Από τα παραπάνω, προκύπτει ότι η πολιτική ασφάλειας μπορεί να ενισχυθεί με τη χρήση ενός συστήματος ανίχνευσης εισβολών. Είναι γεγονός ακόμα, πως η ανάλυση για ασυνήθιστες δραστηριότητες στο δίκτυο γέννησε και την ίδια την ιδέα για συστήματα ανίχνευσης εισβολών [16]. Επιπρόσθετα, η ανίχνευση εισβολών παίζει βασικό ρόλο και στην εγκληματολογική έρευνα, καθώς η καταγραφή του ιστορικού συστημάτων ανίχνευσης εισβολών έχουν κατά καιρούς χρησιμοποιηθεί ως αποδεικτικά στοιχεία σε αντιδικίες [17].

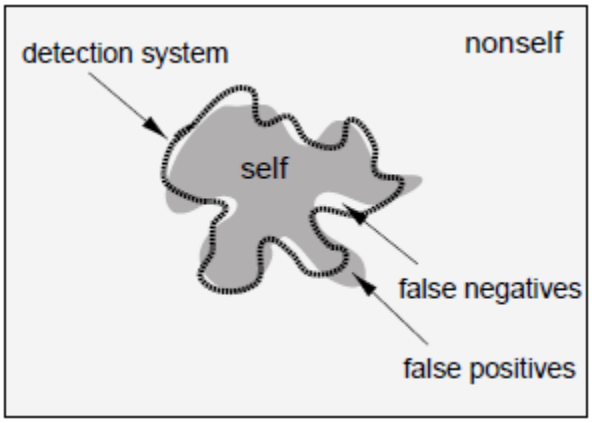
Οι λόγοι για την επιτακτική χρήση συστημάτων ανίχνευσης εισβολών μπορεί να συνοψιστούν στα παρακάτω [5]:

- Αναστολτική δράση, μέσω της αύξησης του ρίσκου αποκάλυψης και επιβολής τιμωρίας στους επίδοξους εισβολείς που ειδάλλως θα παρέκαμπταν ή θα παραβίαζαν την ασφάλεια του συστήματος.

- Η ανίχνευση και η αντιμετώπιση απειλών οι οποίες δεν προλαμβάνονται από τα λοιπά συστήματα ασφάλειας.
- Η ανίχνευση και η αντιμετώπιση ενεργειών που προηγούνται μιας επίθεσης όπως αυτές της βολιδοσκόπησης του συστήματος για ευπάθειες και άλλες δραστηριότητες (“doorknob rattling” activities).
- Η καταγραφή και τεκμηρίωση των υπαρχόντων απειλών σε έναν οργανισμό
- Η δράση ως εργαλείο ελέγχου ποιότητας της σχεδίασης και διαχείρισης ασφάλειας σε ένα, ειδικά μεγάλο, οργανισμό.
- Η παροχή χρήσιμης πληροφορίας σχετικά με τις εισβολές που λαμβάνουν χώρα στο δίκτυο, επιτρέποντας την βελτιωμένη διάγνωση, ανάκαμψη και διόρθωση των αιτίων που δημιούργησαν την εισβολή.

2.3. Στόχοι ενός συστήματος ανίχνευσης εισβολών

Ο βασικός στόχος ενός συστήματος ανίχνευσης εισβολών είναι η διάκριση μεταξύ των μη επιθυμητών και επιθυμητών ενεργειών σε ένα σύστημα ή δίκτυο. Συνεπώς [18], ένα σύστημα ανίχνευσης εισβολών μπορεί να χαρακτηριστεί ως ένα σύστημα ταξινομητής (classifier), το οποίο αναλύει την συμπεριφορά ενός συστήματος ή τα γεγονότα που συμβαίνουν στο σύστημα και να αναγνωρίσει την μοχθηρή συμπεριφορά έναντι των υπολοίπων συμπεριφορών. Στο παρακάτω σχήμα (Σχήμα 1), απεικονίζεται το μοντέλο ενός συστήματος ανίχνευσης εισβολών ως ένα μοντέλο ταξινόμησης (Classification Model) [19].



Σχήμα 1: Η ανίχνευση εισβολών ως ένα σύστημα ταξινομητή (classifier) [19]

Στο σχήμα αυτό, το σκιασμένο κομμάτι χαρακτηρίζει την κανονική συμπεριφορά του συστήματος (self) ενώ όλο το υπόλοιπο την μη αποδεκτή συμπεριφορά. Ένα σύστημα ανίχνευσης εισβολών προσπαθεί να ορίσει την περιοχή μεταξύ των συμπεριφορών αυτών (διακεκομμένη γραμμή). Όπου αποτυγχάνει να αναγνωρίσει τις κανονικές συμπεριφορές, παράγει λανθασμένα θετικά (false positives), ενώ όπου αποτυγχάνει να αναγνωρίσει τις μη αποδεκτές συμπεριφορές, παράγει λανθασμένα αρνητικά (false negatives).

Οι στόχοι ενός συστήματος ανίχνευσης δικτυακών εισβολών, συνοψίζονται στα παρακάτω [7]:

- Να ανιχνεύει μεγάλου εύρους εισβολές, καθώς υπάρχει ανάγκη για εντοπισμό τόσο γνωστών, όσο και άγνωστων εισβολών. Οι εισβολές αυτές μπορεί να προέρχονται είτε από το εξωτερικό του δικτύου, είτε από το εσωτερικό.
- Να ανιχνεύει έγκαιρα τις εισβολές, όχι με την έννοια του πραγματικού χρόνου, αλλά με την έννοια της ανακάλυψης μιας εισβολής σε εύλογο χρονικό διάστημα.

- Να παρουσιάζει την ανάλυση με απλή και εύκολα αντιληπτή μορφή και η διεπαφή με το διαχειριστή ασφάλειας να είναι απλή και κατανοητή. Η απλότητα της ανάλυσης έχει και κάποιο όριο, καθώς πολλές από τις εισβολές δεν είναι πάντα τόσο σαφείς ως προς τη δομή τους.
- Να είναι ακριβής, καθώς είναι ενοχλητικό για τους χρήστες του συστήματος να έχουν προβλήματα στη χρήση του λανθασμένων θετικών (false positives), ενώ από την άλλη είναι επικίνδυνο να μην αναγνωρίζει τις επιθέσεις (false negatives).

Συνοψίζοντας, ένα σύστημα ανίχνευσης εισβολών πρέπει να μεγιστοποιεί το ποσοστό ανίχνευσης εισβολών και να ελαχιστοποιεί το ποσοστό λανθασμένων αρνητικών ειδοποιήσεων, να έχει καλή απόκριση στην ανάλυση των στοιχείων προς ανίχνευση και να είναι λειτουργικά κατανοητό.

2.4. Ταξινόμηση συστημάτων ανίχνευσης εισβολών

Η ανίχνευση εισβολών διακρίνεται σε δύο κύριες κατηγορίες σύμφωνα με την εργασία της Denning [20]: Την ανίχνευση ανωμαλιών (anomaly detection) και στην ανίχνευση υπογραφής (signature detection) η οποία πιο συχνά αναφέρεται ως ανίχνευση κατάχρησης (misuse detection).

2.4.1. Ανίχνευση ανωμαλιών (Anomaly detection)

Η ανίχνευση ανωμαλιών βασίζεται στην στατιστική απόκλιση από την φυσιολογική συμπεριφορά για την ανίχνευση ασυνήθιστης συμπεριφοράς. Η βάση για την φυσιολογική συμπεριφορά του συστήματος γίνεται με βάση την ανάλυση ιστορικών στοιχείων. Η ανάλυση αυτή είναι και η πρώτη φάση ενός συστήματος που βασίζεται σε αυτό το μοντέλο για ανίχνευση εισβολών η οποία ονομάζεται και εκπαίδευση. Η δεύτερη φάση ουσιαστικά είναι η εφαρμογή σε νέα δεδομένα.

Βασικό πλεονέκτημα του παραπάνω μοντέλου είναι η ικανότητα ανίχνευσης νέων ή άγνωστων επιθέσεων, βασιζόμενη στην απόκλιση του συστήματος από την φυσιολογική συμπεριφορά. Επίσης η προσέγγιση αυτή δίνει την δυνατότητα αναγνώρισης παράβασης της πολιτικής ασφάλειας η οποία δεν μπορεί εύκολα να κωδικοποιηθεί.

Μειονέκτημα του μοντέλου αυτού είναι το γεγονός ότι ανιχνεύει ασυνήθιστη συμπεριφορά και όχι απαραίτητα απαγορευμένη. Αυτό έχει ως αποτέλεσμα την πιθανότητα αυξημένων λανθασμένων θετικών προειδοποιήσεων (false positives alerts) για γεγονότα τα οποία είναι φυσιολογικά, αποκλίνουν όμως από την καταγεγραμμένη φυσιολογική δυναμική του συστήματος. Ομοίως, επιθέσεις των οποίων το αποτύπωμά τους βασίζεται σε αυτό της φυσιολογικής συμπεριφοράς, δεν θα ανιχνευθεί.

Για την αντιμετώπιση των παραπάνω, το σύστημα εκπαιδεύεται σε τακτά χρονικά διαστήματα από τα δεδομένα που καταγράφονται ώστε να ενημερώνεται κάθε φορά η φυσιολογική συμπεριφορά. Αυτό βέβαια είναι χρονοβόρο και απαιτεί καλή ανάλυση των καταγεγραμμένων στοιχείων από το δίκτυο.

Η ανίχνευση ανωμαλιών, ανάλογα με τη μέθοδο υλοποίησης, μπορεί να κατηγοριοποιηθεί στις εξής τρεις [21]:

- Ανίχνευση ανωμαλιών με βάση στατιστικές μεθόδους,
- Ανίχνευση ανωμαλιών με βάση τη μηχανική μάθησης,
- Ανίχνευση ανωμαλιών με βάση την εξόρυξη δεδομένων.

Πολύ συχνά στην πραγματικότητα χρησιμοποιείται συνδυασμός των παραπάνω τεχνικών, έτσι ώστε να προκύπτουν υβριδικά συστήματα ανίχνευσης ανωμαλιών.

2.4.2. Ανίχνευση κατάχρησης (Misuse detection)

Η ανίχνευση κατάχρησης βασίζεται στην κωδικοποίηση γνωστών προφίλ εκ των προτέρων στη βάση δεδομένων του συστήματος ανίχνευσης. Χρησιμοποιώντας προφίλ γνωστών εισβολών μπορεί μέσω

σύγκρισης αλφαριθμητικών ή αναγνώριση προτύπων (string or pattern matching), να αναγνώριση των εισβολών.

Το πλεονέκτημα του μοντέλου αυτού είναι η ικανότητα του να ανιχνεύει με ακρίβεια γνώριμες επιθέσεις. Βασικό όμως μειονέκτημα είναι η αδυναμία αναγνώρισης νέων ή άγνωστων εισβολών που δεν υπάρχουν στη συλλογή με τις υπάρχουσες.

Για την αντιμετώπιση του γεγονότος αυτού, η συλλογή με τις εισβολές ενημερώνεται σε τακτά χρονικά διαστήματα με νέες εισβολές. Η διαδικασία αυτή χρειάζεται προσεκτική ανάλυση του δικτύου και μπορεί να είναι χρονοβόρα ή να έχει κόστος, στην περίπτωση που αγοράζονται νέες λίστες με εισβολές από προμηθευτή.

Παρόλα αυτά η ανίχνευση κατάχρησης αποτελεί την πιο ευρέως χρησιμοποιούμενη τεχνική στα συστήματα ανίχνευσης εισβολών. Ένας λόγος για αυτό είναι και το λογισμικό ανίχνευσης εισβολών Snort [22], το οποίο διατίθεται ελεύθερα.

Η ανίχνευση κατάχρησης, ανάλογα με τη μέθοδο υλοποίησης της, μπορεί να ταξινομηθεί στις εξής τρεις κύριες κατηγορίες [23]:

- Ανίχνευση κατάχρησης με χρήση έμπειρων συστημάτων,
- Ανίχνευση κατάχρησης με μεθόδους αναγνώρισης προτύπων,
- Ανίχνευση κατάχρησης με μεθόδους ανάλυσης μετάβασης καταστάσεων.

2.4.3. Ανίχνευση με βάση προδιαγραφές (Specification-based detection)

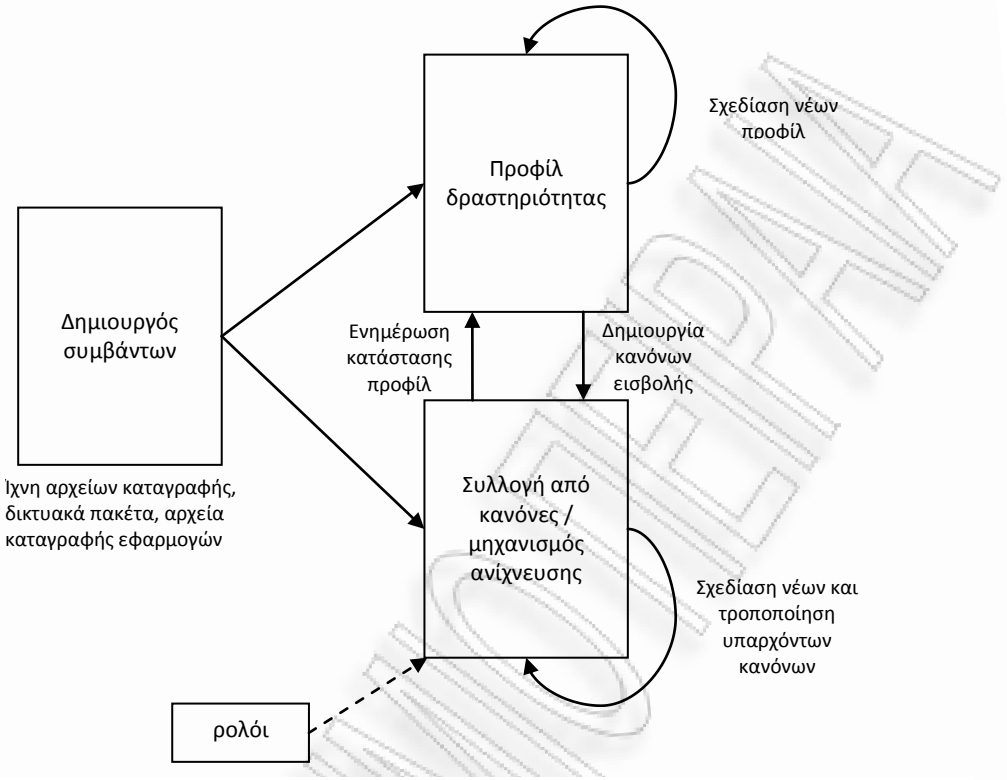
Μια πρόσφατη κατηγορία ανίχνευσης είναι η ανίχνευση με βάση προδιαγραφές [24]. Το μοντέλο αυτό είναι παρόμοιο με αυτό της ανίχνευσης ανωμαλιών στο ό,τι και τα δύο μοντέλα ανιχνεύουν αποκλίσεις από αυτό που θεωρείται φυσιολογικό. Η διαφορά τους έγκειται στο ότι στην ανίχνευση ανωμαλιών πραγματοποιείται εκπαίδευση συνήθως με κάποια τεχνική μηχανικής εκμάθησης, ενώ στην ανίχνευση προδιαγραφών βασίζεται σε χειροκίνητες προδιαγραφές οι οποίες περιγράφουν τις φυσιολογικές συμπεριφορές του συστήματος.

Το πλεονέκτημα στη μέθοδο αυτή είναι η αποφυγή αρκετών λανθασμένων θετικών αλλά από την άλλη η χειροκίνητη παραμετροποίηση των προδιαγραφών αποβαίνει αρκετά χρονοβόρα [25].

2.5. Μοντέλο συστημάτων ανίχνευσης εισβολών

Ένα γενικό μοντέλο για την ανίχνευση εισβολών εισήχθη αρχικά από την Dorothy Denning [20] και το μοντέλο αυτό συνεχίζει να ισχύει και σήμερα σαν γενικότερη προσέγγιση ενός μηχανισμού ανίχνευσης εισβολών [15].

Το μοντέλο αυτό (Σχήμα 2) έχει τρεις βασικές συνιστώσες, τον δημιουργό συμβάντων (event generator), το προφίλ δραστηριότητας (activity profile) και τη συλλογή κανόνων (rule set). Ο δημιουργός συμβάντων αποτελεί το μέσο από το οποίο τροφοδοτούνται δεδομένα ώστε να παραχθεί πληροφορία για τη δραστηριότητα που συμβαίνει. Τα δεδομένα αυτά προέρχονται από πλήθος ειδών καταγραφής, ίχνη καταγραφής από λειτουργικά συστήματα, από καταγραφή της κίνησης του δικτύου, από εφαρμογές λογισμικού και από άλλα συστήματα ασφαλείας όπως από αναχώματα ασφάλειας και λίστες πρόσβασης.



Σχήμα 2: Ένα γενικό μοντέλο ανίχνευσης εισβολών

Η συλλογή από κανόνες είναι ο μηχανισμός που ουσιαστικά παράγει συμπεράσματα για το αν μια εισβολή έχει λάβει χώρα ή όχι. Στη συνιστώσα αυτή, θεωρείται ότι λαμβάνει χώρα η ανάλυση με βάση μοντέλα, στατιστικά στοιχεία, πρότυπα και όποια άλλη μεθοδολογία έχει υλοποιηθεί για την ανίχνευση εισβολών.

Το προφίλ δραστηριότητας είναι η συνιστώσα η οποία διατηρεί την κατάσταση του συστήματος/δικτύου που παρακολουθείται. Τροφοδοτείται από το δημιουργό συμβάντων και ανάλογα, αλλάζει τις μεταβλητές που συνιστούν το προφίλ του δικτύου. Το προφίλ δραστησριότητας αλλάζει επίσης μεταβλητές του προφίλ, με βάση την τροφοδότηση που έχει από το μηχανισμό ανίχνευσης, ανάλογα με τις ενέργειες που ο μηχανισμός ανίχνευσης επιτελεί.

Το ρολόι, ως συνιστώσα αντιπροσωπεύει το χρονικό διάστημα στο οποίο θα υποδεικνύεται στο μηχανισμό ανίχνευσης για την εκπαίδευση και την πιθανή παραγωγή νέων κανόνων.

Κάθε συνιστώσα, ανάλογα με την υλοποίηση μπορεί να τοποθετηθεί σε διαφορετικά σημεία στο δίκτυο που παρακολουθείται.

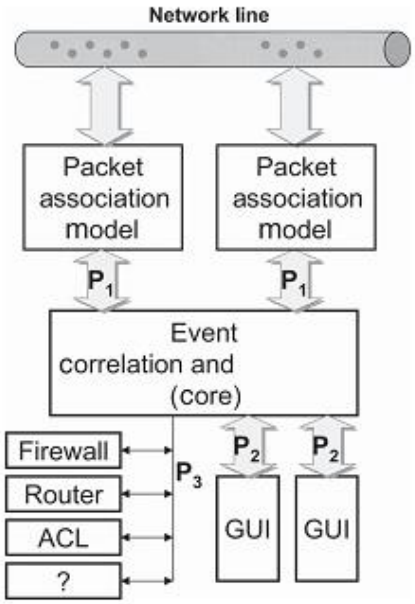
Στο Σχήμα 3, παρουσιάζεται μια προσέγγιση ενός σύγχρονου συστήματος ανίχνευσης δικτυακών εισβολών και η τοποθέτησή του στα πλαίσια ενός δικτύου σε συνεργασία με άλλα συστήματα ασφάλειας. Οι συνιστώσες του μοντέλου αυτού είναι το Μοντέλο συσχέτισης πακέτων (Packet Association Model), ο Πυρήνας Συσχέτισης και Ανίχνευσης συμβάντων (Event Correlation and Detection Core) και μια διεπαφή χρήστη (Graphical User Interface – GUI).

Το μοντέλο συσχέτισης πακέτων αποκαλείται και ως αισθητήρας του συστήματος ανίχνευσης εισβολών. Σε μεγάλες εφαρμογές, συνήθως διαθέτει δυνατότητες επεξεργασίας και μπορεί να εκτελεί από φιλτράρισμα και εξαγωγή μεταδεδομένων έως πολύπλοκες στατιστικές αναλύσεις.

Ο πυρήνας συσχέτισης και ανίχνευσης συμβάντων ουσιαστικά αποτελεί τη συλλογή των παραγόμενων συσχετίσεων πακέτων δεδομένων (κανόνες).

Η διεπαφή με το χρήστη, ουσιαστικά με το διαχειριστή ασφάλειας, παρέχει συνήθως με γραφικό τρόπο, την έξοδο των αποτελεσμάτων και των γεγονότων του συστήματος ανίχνευσης από πολλαπλά σημεία.

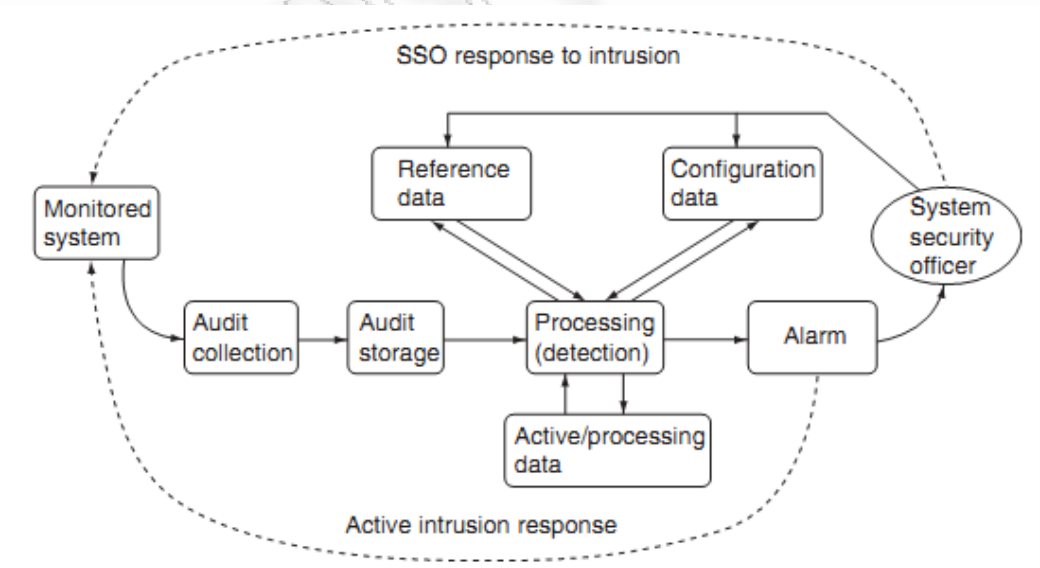
Η επικοινωνία μεταξύ των υποσυστημάτων (P_1, P_2) γίνεται με διάφορα πρωτόκολλα επικοινωνίας. Η επικοινωνία μεταξύ πυρήνα και μοντέλου συσχέτισης πακέτων γίνεται συνήθως με βάση το SNMP πρωτόκολλο (Simple Network Management Protocol), ενώ η επικοινωνία μεταξύ πυρήνα συσχέτισης - ανίχνευσης συμβάντων και των διεπαφών χρήστη με διαδικτυακά πρωτόκολλα (HTTP) ή πρωτόκολλα TCP (Transmission Control Protocol).



Σχήμα 3: Μια προσέγγιση ενός σύγχρονου συστήματος ανίχνευσης δικτυακών εισβολών [13]

Όσον αφορά στην επικοινωνία με τρίτα συστήματα ασφαλείας (P_3), αυτή εξαρτάται συνήθως από τους προμηθευτές συστημάτων ασφάλειας (αναχώματα ασφαλείας, δρομολογητές, λίστες ελέγχου πρόσβασης κ.α.) και μπορεί να είναι διαφόρων τύπων.

Το παραπάνω μοντέλο μπορεί να εκφραστεί όσον αφορά στις λειτουργικές του συνιστώσες όπως το Σχήμα 4 [26].



Σχήμα 4: Λειτουργικά στοιχεία ενός τυπικού συστήματος ανίχνευσης εισβολών [26]

Το κεντρικό σημείο του συστήματος είναι η επεξεργασία/ανίχνευση (Processing/detection). η επεξεργασία/ανίχνευση τροφοδοτείται από το σύστημα με τα διάφορα γεγονότα που συμβαίνουν

(Audit collection, Audit Storage). Επίσης, τροφοδοτείται από τα δεδομένα παραμετροποίησης (Configuration data) αλλά και από πρότυπα δεδομένα (Reference data). Αυτά με τη σειρά τους καθορίζονται και ελέγχονται από τον διαχειριστή ασφάλειας του συστήματος/δικτύου. Η επεξεργασία επίσης, χρειάζεται ένα είδους ενδιάμεσης αποθήκευσης (cache) για τα δεδομένα που έχει να επεξεργαστεί (Active/processing data).

Η έξοδος από την επεξεργασία/ανίχνευση συνήθως έχει τη μορφή μιας προειδοποίησης η οποία μεταδίδεται αφενός μεν στον διαχειριστή ασφάλειας ώστε να κάνει ενέργειες, αφετέρου δε μπορεί να δρα αυτόνομα στο δίκτυο και να εκτελεί ορισμένες αυτοματοποιημένες ενέργειες. Στην περίπτωση που δρα αυτόνομα το σύστημα μπορεί να εκφραστεί ως συνδυασμός ανίχνευσης και αποτροπής εισβολών (Intrusion detection and prevention).

3. Γενετικοί Αλγόριθμοι

Στην ενότητα αυτή, γίνεται μια συνοπτική παρουσίαση των γενετικών αλγορίθμων, ξεκινώντας από τον προσδιορισμό τους, την προέλευσή τους, την εξέλιξή τους, το πεδίο εφαρμογής τους, τα χαρακτηριστικά τους αλλά και τα πλεονεκτήματά τους σε σχέση με τις άλλες μεθόδους, καθώς και γιατί δουλεύουν σωστά αλλά και ποιά είναι τα προβλήματά τους και για ποιο λόγο τους έχει ασκηθεί κριτική. Στο τέλος γίνεται και μια σύντομη αναφορά σε βασικές εφαρμογές των Γενετικών Αλγορίθμων.

3.1. Εισαγωγή – Τι είναι οι Γενετικοί Αλγόριθμοι

Τις τελευταίες τρεις δεκαετίες, το ενδιαφέρον της ερευνητικής και επιστημονικής κοινότητας παρουσιάζει ένα αυξανόμενο ενδιαφέρον για εναλλακτικές μεθόδους επίλυσης προβλημάτων. Μια από τις βασικές αιτίες είναι η αδυναμία που παρουσιάζουν οι κλασσικές μέθοδοι για επίλυση μεγάλου αριθμού προβλημάτων και τα οποία προβλήματα δεν μπορούν πάντα να ξεπεραστούν με υβριδικές μεθόδους ή πολλές φορές οι υβριδικές μέθοδοι είναι μη αποδοτικές. Επίσης, άλλη μια αιτία είναι και η παρουσίαση ολοένα και δυσκολότερων προβλημάτων, γεγονός το οποίο κάνει επιτακτική την ανάγκη για εξεύρεση νέων μεθόδων αναζήτησης και επίλυσης αυτών. Απ' την άλλη, ο στόχος της δημιουργίας τεχνητής ευφυΐας υπήρχε από την αρχή της δημιουργίας του υπολογιστή [27]. Πολλοί από τους πρώτους επιστήμονες όπως ο Alan Turing, ο John von Neumann και άλλοι προσπαθούσαν να εμπλουτίσουν τα προγράμματά και τις κατασκευές τους με ευφυΐα αλλά και με την ικανότητα να μαθαίνουν και να προσαρμόζονται ανάλογα με το πεδίο δράσης τους. Ως εκ τούτου, από νωρίς στράφηκε το ενδιαφέρον, προς τη μοντελοποίηση του εγκεφάλου, τη μίμηση της ανθρώπινης εκμάθησης και την προσομοίωση της βιολογικής εξέλιξης. Αυτοί οι τρεις στόχοι εξελίχθηκαν αντίστοιχα στα πεδία των Νευρωνικών Δικτύων, την Μηχανική Εκμάθησης και σε αυτό που σήμερα ονομάζουμε Εξελικτικό Προγραμματισμό (Evolutionary Computation), χαρακτηριστικό παράδειγμα του οποίου είναι οι Γενετικοί Αλγόριθμοι.

3.1.1. Η θεωρία της Εξέλιξης των Ειδών

Η θεωρία της Εξέλιξης των Ειδών (Evolution of Species) αναπτύχθηκε στα μέσα του 19^{ου} αιώνα από τον Δαρβίνο. Μαζί με την διατύπωσή της ήρθε και ένα κύμα αναστάτωσης καθώς η φιλοσοφία της ερχόταν σε σύγκρουση με τις επικρατούσες θρησκευτικές πεποιθήσεις για την προέλευση της ζωής. Τον επόμενο αιώνα, η θεωρία άρχισε να γίνεται αποδεκτή από ένα ευρύ φάσμα της επιστημονικής κοινότητας, καθώς η θεωρία έδινε ικανοποιητικές απαντήσεις σε θεμελιώδη ερωτήματα [28]. Ο σκοπός της θεωρίας του Δαρβίνου είναι να ερμηνεύσει την προέλευση και τις βασικές λειτουργίες του φαινομένου της ζωής. Τα κυριότερα στοιχεία της, τα οποία σχετίζονται και άμεσα με τον τρόπο λειτουργίας των Γενετικών Αλγορίθμων είναι τα παρακάτω:

- Δεν υπάρχει αντικειμενική βάση διαχωρισμού των ζωντανών οργανισμών σε ανώτερους και κατώτερους (όσον αφορά το ίδιο βιολογικό είδος όπως π.χ. ο άνθρωπος). Σε κάθε βιολογικό είδος, ορισμένα άτομα αφήνουν περισσότερους απογόνους σε σχέση με τα υπόλοιπα συνεπώς τα κληροδοτούμενα χαρακτηριστικά των αναπαραγωγικά επιτυχημένων ατόμων γίνονται περισσότερα στην επόμενη γενιά. Οι δυσκολίες και τα εμπόδια του περιβάλλοντος είναι οι παράγοντες που καθορίζουν ποιοι από τους οργανισμούς θα κατορθώσουν να επιβιώσουν και να πολλαπλασιαστούν. Έτσι και οι οργανισμοί αλλάζουν τα χαρακτηριστικά τους προσπαθώντας να προσαρμοστούν, έτσι ώστε να επιβιώσουν.
- Η αλλαγή που επέρχεται στα χαρακτηριστικά των οργανισμών, είναι αλλαγή στα *χρωμοσώματά* τους *(chromosomes)*, που είναι πολύπλοκα οργανικά μόρια που κωδικοποιούν τη δομή και τα χαρακτηριστικά τους. Αυτά, με τη σειρά τους, αποτελούνται από μικρότερα μέρη, τα *γονίδια (genes)*, ενώ το σύνολο της πληροφορίας που είναι κωδικοποιημένο στα γονίδια ονομάζεται *γονότυπος (genotype)*. Η δημιουργία ενός νέου οργανισμού αντιστοιχεί στην αποκωδικοποίηση των χρωμοσωμάτων. Το σύνολο των

«ορατών» χαρακτηριστικών (π.χ. χρώμα ματιών) και η συμπεριφορά (π.χ. εξυπνάδα) ενός οργανισμού αποτελούν τον *φαινότυπο (phenotype)*.

- Οι κυρίαρχες λειτουργίες του φαινομένου της εξέλιξης είναι η αναπαραγωγή (reproduction) και η μετάλλαξη *(mutation)*. Στη μετάλλαξη συμβαίνει η τυχαία αλλαγή της δομής των χρωμοσωμάτων είτε από λανθασμένη αντιγραφή βιολογικών μορίων είτε από εξωγενείς παράγοντες (π.χ. ακτινοβολία). Η μετάλλαξη ορισμένες φορές, μπορεί να προκαλέσει βελτίωση στα χαρακτηριστικά ενός οργανισμού και να αποτελέσει παράγοντα θετικής εξέλιξης της ζωής.
- Προϊόν της αναπαραγωγής είναι ένας νέος οργανισμός, τα χρωμοσώματα του οποίου αποτελούνται από γονίδια που προέρχονται τα μισά από τον πατέρα και τα μισά από τη μητέρα, δηλαδή για κάθε χαρακτηριστικό, το νέο άτομο θα έχει πάρει ένα γονίδιο από κάθε γονέα. Αυτό που συμβαίνει για τα δυο γονίδια είναι είτε να συμφωνούν ως προς την «τιμή» που θα δώσουν στο χαρακτηριστικό (π.χ. και τα δύο γαλάζιο χρώμα ματιών), είτε να μη συμφωνούν. Στην τελευταία περίπτωση, κυριαρχεί η «τιμή» του ενός και αγνοείται του άλλου, αλλά και του άλλου μπορεί να περάσει σε επόμενες γενιές. Το γονίδιο που καθορίζει τελικά το χαρακτηριστικό λέγεται *επικρατές (dominant)* και το άλλο *υπολειπόμενο (recessive)*. Τα γονίδια που διεκδικούν την ίδια θέση σε ένα χρωμόσωμα (είναι και τα δυο υπεύθυνα για το ίδιο χαρακτηριστικό) λέγονται *αλληλόμορφα (alleles)*.

3.1.2. Η γένεση των Γενετικών Αλγορίθμων

Όλη η φιλοσοφία της θεωρίας εξέλιξης των ειδών, ο μηχανισμός δηλαδή της φυσικής επιλογής, φάνηκε ενδιαφέρον στον John Holland, ο οποίος μαζί με τους συνεργάτες του από το Πανεπιστήμιο του Michigan ανέπτυξε τους Γενετικούς Αλγόριθμους στις αρχές του 1970 [29]. Ο στόχος της έρευνάς του στο αντικείμενο αυτό ήταν διπλός: Αφενός να μοντελοποιήσει και να εξηγήσει με αυστηρό τρόπο την διαδικασία προσαρμογής των φυσικών συστημάτων (natural systems) και αφετέρου να σχεδιάσει τεχνητά συστήματα λογισμικού τα οποία διατηρούν τους βασικότερους μηχανισμούς των φυσικών συστημάτων. Το αποτέλεσμα αυτής της προσέγγισης της έρευνας οδήγησε στους Γενετικούς Αλγόριθμους.

Η βασική ιδέα των Γενετικών Αλγορίθμων είναι η μίμηση των μηχανισμών της φύσης. Για παράδειγμα αν πάρουμε ένα παράδειγμα από ένα συγκεκριμένο θαλάσσιο είδος π.χ. τους ροφούς οι οποίοι αναπαράγονται και εξελίσσονται από γενιά σε γενιά. Ορισμένοι από αυτούς είναι εξυπνότεροι και πιο γρήγοροι από τους υπόλοιπους με αποτέλεσμα να έχουν μεγαλύτερη πιθανότητα επιβίωσης από εχθρούς τους όπως π.χ. οι ψαροντουφεκάδες. Απ την άλλη όμως θα υπάρχουν και ορισμένοι ροφοί όχι τόσο γρήγοροι ή εύστροφοι, οι οποίοι θα επιβιώσουν και αυτοί με τη σειρά τους λόγω π.χ. τύχης. Όλοι αυτοί που θα γλιτώσουν θα αναπαραχθούν μεταξύ τους και θα ξεκινήσει η παραγωγή της επόμενης γενιάς, η οποία θα συνδυάζει διάφορα και ποικίλα χαρακτηριστικά από την προηγούμενη καθώς ορισμένοι γρήγοροι ροφοί θα διασταυρωθούν με ορισμένους πιο αργούς ή λιγότερο εύστροφους, άλλοι γρήγοροι με άλλους γρήγορους, άλλοι εύστροφοι με λιγότερο η μη εύστροφους και ούτω καθεξής. Το αποτέλεσμα από αυτή τη διαδικασία είναι ότι οι επόμενοι ροφοί που θα γεννηθούν θα είναι κατά μέσο όρο, γρηγορότεροι και εξυπνότεροι από τους προγόνους τους εφόσον από την προηγούμενη γενιά επιβίωσαν περισσότεροι γρήγοροι και έξυπνοι ροφοί.

Με τον ίδιο τρόπο και οι Γενετικοί Αλγόριθμοι προσομοιώνουν την φυσική διαδικασία: Συνδυάζουν τα καλύτερα χαρακτηριστικά των ατόμων, μεταφρασμένα σε δομές συμβολοσειρών, και τα αναπαράγουν ώστε να δημιουργηθούν ακόμα καλύτερα στην επόμενη γενιά έως ότου βρεθεί η βέλτιστη λύση. Μπορεί οι διαδικασίες αναπαραγωγής και επιλογής των συμβολοσειρών – ατόμων να βασίζονται στην τύχη, όπως και στη φύση, αλλά δεν είναι βάδισμα στην τύχη: Εκμεταλλεύονται παρελθούσα πληροφορία ώστε να βαδίσουν σε σημεία του χώρου αναζήτησης τα οποία υποθέτουν ότι είναι βελτιωμένα.

3.2. Η δομή των Γενετικών Αλγορίθμων

Στην ενότητα αυτή περιγράφεται ο τρόπος με τον οποίο τα στοιχεία της θεωρίας της Εξέλιξης χρησιμοποιούνται στη δόμηση ενός Γενετικού Αλγόριθμου, η αντιστοίχιση δηλαδή των φυσικών χαρακτηριστικών σε δομικά στοιχεία τα οποία μπορούν να εφαρμοστούν σε πρόβλημα βελτιστοποίησης.

Βασικό δομικό στοιχείο για ένα Γενετικό Αλγόριθμο είναι το χρωμόσωμα. Σε αντίθεση με τους φυσικούς οργανισμούς οι οποίοι μπορεί να έχουν πολλά χρωμοσώματα (τα ανθρώπινα κύτταρα παράδειγμα έχουν 46 χρωμοσώματα), στους Γενετικούς Αλγόριθμους σχεδόν πάντα αναφερόμαστε σε άτομα με ένα χρωμόσωμα. Τα γονίδια ενός Γενετικού Αλγόριθμου είναι διατεταγμένα σε γραμμική ακολουθία και βρίσκονται σε συγκεκριμένες θέσεις (locus) στο χρωμόσωμα. Τα αλληλόμορφα (alleles) στην περίπτωση του Γενετικού Αλγόριθμου είναι οι διαφορετικές τιμές που μπορεί να πάρει το κάθε γονίδιο. Αν για παράδειγμα χρησιμοποιούμε Γενετικό με γονίδια δυαδικά, οι τιμές που μπορεί να πάρει το κάθε γονίδιο είναι 0 και 1. Επίσης κάθε γονότυπος (χρωμόσωμα) αναπαριστά μια πιθανή λύση στο πρόβλημα. Η μετάφραση αυτού του περιεχομένου του χρωμοσώματος καλείται φαινότυπος και εξαρτάται άμεσα από τον σχεδιαστή. Η αναζήτηση λύσης σε ένα Γενετικό Αλγόριθμο γίνεται διατηρώντας ένα αρχικό πληθυσμό από πιθανές λύσεις και οι οποίες υφίστανται μια προσομοιωμένη γενετική εξέλιξη. Ο διαχωρισμός και η αξιολόγηση των διαφόρων λύσεων, που παράγονται σε κάθε γενιά, γίνεται μέσω μιας αντικειμενικής συνάρτησης (fitness function) ή μιας συνάρτησης ικανότητας, καταλληλότητας, αξιολόγησης. Την αξιολόγηση των λύσεων ακολουθεί η επιλογή των καλύτερων λύσεων και η εφαρμογή των γενετικών τελεστών της διασταύρωσης και μετάλλαξης.

Συνοψίζοντας τα παραπάνω, τα τμήματα τα οποία συντελούν στη δόμηση και τη λειτουργία ενός Γενετικού Αλγόριθμου για ένα συγκεκριμένο πρόβλημα είναι τα παρακάτω:

1. Μια γενετική αναπαράσταση των πιθανών λύσεων του προβλήματος. Για παράδειγμα έχουμε ένα πληθυσμό $P(t) = \{X_1^t, \ldots, X_n^t\}$, όπου με t συμβολίζουμε την επαναληπτική εκτέλεση.
2. Έναν τρόπο δημιουργίας ενός αρχικού πληθυσμού. Συνήθως η αρχική δημιουργία ενός πληθυσμού γίνεται με τυχαίο τρόπο.
3. Μια αντικειμενική συνάρτηση αξιολόγησης η οποία παίζει το ρόλο του περιβάλλοντος. Με τη συνάρτηση αυτή αξιολογούνται όλες οι πιθανές λύσεις του πληθυσμού, δίνοντας τους τιμές με τις οποίες θα μπορεί να γίνει κατάταξη και σύγκριση ώστε να ξεχωρίσουν οι καλύτερες. Αφού επιλεχθούν τα καταλληλότερα δημιουργείται νέος πληθυσμός με επαναληπτική εκτέλεση t+1.
4. Γενετικούς τελεστές που μετατρέπουν τη σύνθεση των παιδιών. Ορισμένα μέλη από τον καινούριο πληθυσμό υφίστανται τους γενετικούς τελεστές της διασταύρωσης και της μετάλλαξης και σχηματίζουν νέες υποψήφιες λύσεις.
5. Τιμές για διάφορες παραμέτρους που χρησιμοποιεί ο Γενετικός Αλγόριθμος και σχετίζονται με το μέγεθος του πληθυσμού, πιθανότητες εφαρμογής γενετικών τελεστών και άλλες.

3.3. Γιατί χρησιμοποιούνται οι Γενετικοί Αλγόριθμοι

3.3.1. Πλεονεκτήματα των Γενετικών Αλγορίθμων

Οι Γενετικοί Αλγόριθμοι διαθέτουν ένα πλήθος από χαρακτηριστικά τα οποία δημιουργούν τις κατάλληλες προϋποθέσεις για την αξιοποίησή τους στην επίλυση των προβλημάτων. Βασικά πλεονεκτήματα αποτελούν τα παρακάτω [28]:

1. Οι Γενετικοί Αλγόριθμοι μπορούν να επιλύουν δύσκολα προβλήματα γρήγορα και αξιόπιστα καθώς ένα βασικό χαρακτηριστικό τους είναι η μεγάλη αποδοτικότητά τους. Αυτό απορρέει τόσο από τη θεωρία όσο και από την πράξη όπου έχουν αντιμετωπιστεί προβλήματα τα οποία έχουν πολλές και δύσκολα προσδιορισμένες λύσεις.

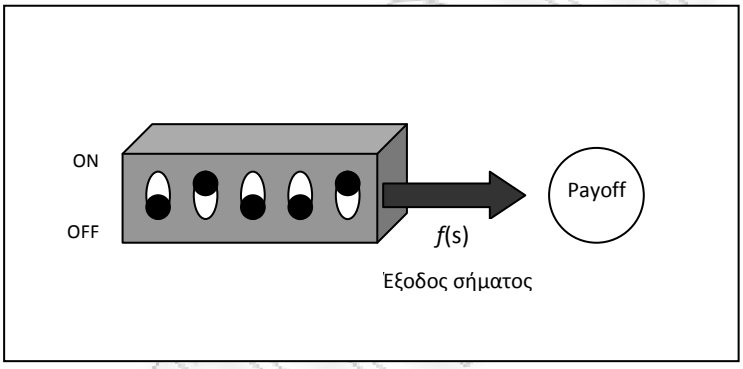
2. Συνεργάζονται εύκολα με υπάρχοντα μοντέλα και συστήματα γιατί μπορούν να χρησιμοποιηθούν με προσθετικό τρόπο στα μοντέλα που ήδη χρησιμοποιούνται χωρίς να απαιτείται η επανασχεδίασή τους. Έτσι μπορούν να συνεργαστούν με υπάρχοντα κώδικα με μικρό κόστος. Αυτό συμβαίνει γιατί χρησιμοποιούν μόνο πληροφορίες οι οποίες σχετίζονται με τη συνάρτηση που πρόκειται να βελτιστοποιήσουν, χωρίς να παίζει άμεσα ρόλο η όλη δομή του συστήματος.
3. Είναι εύκολα επεκτάσιμοι και εξελίξιμοι. Σε πολλές εφαρμογές υπάρχει η δυνατότητα ή και ακόμα επιβάλλεται να χρησιμοποιηθούν λειτουργίες οι οποίες δεν είναι αντιγραμμένες από τη φύση ή λειτουργίες αντιγραμμένες από τη φύση και οι οποίες έχουν υποστεί σημαντικές αλλαγές προς όφελος της απόδοσης.
4. Ως αποτέλεσμα της ευελιξίας τους, υπάρχει η δυνατότητα συμμετοχής σε υβριδικές μορφές με άλλες μεθόδους. Σε συγκεκριμένα προβλήματα όπου άλλες μέθοδοι έχουν υψηλή αποδοτικότητα λόγω εξειδίκευσης, οι Γενετικοί Αλγόριθμοι μπορεί να χρησιμοποιηθούν σε συνδυασμό με τις άλλες μεθόδους.
5. Μπορούν να εφαρμοστούν σε πολλά πεδία προβλημάτων. Η ελευθερία επιλογής των κριτηρίων που καθορίζουν την επιλογή μέσα σύστημα-περιβάλλον τους καθιστά ικανούς για να χρησιμοποιηθούν στην οικονομία, στο σχεδιασμό μηχανών, στην επίλυση μαθηματικών εξισώσεων, στην εκπαίδευση Νευρωνικών Δικτύων και σε πληθώρα άλλων εφαρμογών.
6. Δεν απαιτούν περιορισμούς στις συναρτήσεις που επεξεργάζονται. Πολλές από τις παραδοσιακές μεθόδους απαιτούν περιορισμούς αλλά ενέχουν και πολύπλοκες ιδιότητες για τις οποίες ένας Γενετικός Αλγόριθμος είναι αδιάφορος. Αυτό το γεγονός αυξάνει την δυνατότητα των Γενετικών Αλγορίθμων να επιλύουν ένα μεγάλο φάσμα προβλημάτων.
7. Δεν ενδιαφέρει η σημασία της υπό εξέταση πληροφορίας. Η μόνη επικοινωνία του Γενετικού Αλγόριθμου με το περιβάλλον του είναι η αντικειμενική συνάρτηση η οποία δίνει λύσεις ανεξάρτητα από τη σημασία του προβλήματος. Έτσι, στα προβλήματα που δεν μπορεί να δώσει λύση ένας Γενετικός Αλγόριθμος, αιτία είναι η φύση του χώρου που ερευνά και όχι το πληροφοριακό περιεχόμενο των προβλημάτων αυτών.
8. Έχουν από τη φύση τους το στοιχείο του παραλληλισμού μιας και σε κάθε τους βήμα επεξεργάζονται μεγάλες ποσότητες πληροφορίας και κάθε άτομο στον πληθυσμό αντιπροσωπεύει πολλά άλλα (έχει υπολογιστεί ότι η αναλογία αυτή είναι της τάξης $O(n^3)$, δηλαδή 10 άτομα αντιπροσωπεύουν περίπου 1000). Συνεπώς μπορούν να κάνουν αποδοτική αναζήτηση μεγάλων χώρων σε μικρούς χρόνους.
9. Συνδυάζουν ταυτόχρονη εξερεύνηση του χώρου αναζήτησης και εκμετάλλευση της ήδη επεξεργασμένης πληροφορίας, συνδυασμός που σπάνια συναντάται σε άλλη μέθοδο. Συνήθως γίνεται καλή εξερεύνηση του χώρου αλλά δεν γίνεται εκμετάλλευση της πληροφορίας ή γίνεται καλή εκμετάλλευση πληροφορίας (π.χ. αναζήτηση με μικρά άλματα – hillclimbing) χωρίς να γίνεται καλή εξερεύνηση του χώρου αναζήτησης. Η συνύπαρξη και των δύο χαρακτηριστικών είναι δύσκολη καθώς είναι ανταγωνιστικά μεταξύ τους, γεγονός που προσδίδει μεγαλύτερη αποδοτικότητα στους Γενετικούς Αλγόριθμους.
10. Λόγω της φύσης τους, επιδέχονται παράλληλη υλοποίηση αξιοποιώντας, έτσι, τα πλεονεκτήματα των παράλληλων μηχανών.

3.3.2. Διαφορές Γενετικών Αλγορίθμων από κλασσικές μεθόδους

Τα πλεονεκτήματα των Γενετικών Αλγορίθμων απορρέουν από το σύνολο των χαρακτηριστικών που διαθέτουν και τους κάνει να διαφέρουν από τις κλασσικές μεθόδους. Οι θεμελιώδεις διαφορές είναι οι εξής παρακάτω [30]:

1. Οι Γενετικοί Αλγόριθμοι δουλεύουν κωδικοποιώντας το σύνολο τιμών των παραμέτρων – μεταβλητών και όχι με τις παραμέτρους αυτές καθ’ αυτές. Οι Γενετικοί Αλγόριθμοι απαιτούν το σύνολο των φυσικών παραμέτρων του προβλήματος να κωδικοποιηθεί ως μια

πεπερασμένου μήκους συμβολοσειρά χρησιμοποιώντας ένα πεπερασμένο αλφάβητο. Για παράδειγμα (Σχήμα 5), αν υποθέσουμε ότι έχουμε ένα μαύρο κουτί με πέντε δυαδικούς διακόπτες (on-off). Για κάθε συνδυασμό των διακοπτών s παράγεται μια έξοδος *f*(s). Το ζητούμενο του προβλήματος είναι να τεθούν με τέτοιο τρόπο οι διακόπτες ώστε να παραχθεί η μέγιστη δυνατή τιμή της *f*. Με τις περισσότερες κλασσικές μεθόδους, συνήθως θα δουλεύαμε απευθείας με τις μεταβλητές (την θέση των διακοπτών on ή off) ανοιγοκλείνοντας διακόπτες και χρησιμοποιώντας τους κανόνες μετάβασης της εκάστοτε μεθόδου. Με τους Γενετικούς Αλγόριθμους, αρχικά κωδικοποιούμε τους διακόπτες σε ένα πεπερασμένου μήκους αλφαριθμητικό. Στο απλό αυτό παράδειγμα μπορούμε να αντιστοιχίσουμε 1 στην περίπτωση που ένας διακόπτης είναι ανοιχτός και 0 στην περίπτωση που είναι κλειστός. Με την κωδικοποίηση αυτή, η συμβολοσειρά 11110 αναπαριστά τον τρόπο τοποθέτησης των διακοπτών οπού οι 4 πρώτοι διακόπτες είναι ανοιχτοί και ο τελευταίος κλειστός. Ο τρόπος που θα κωδικοποιήσουμε κάθε φορά τις παραμέτρους, δυαδική, δεκαδική, GRAY κ.τ.λ, εξαρτάται από το πρόβλημα που μελετάμε κάθε φορά και σε πολλές περιπτώσεις δεν είναι προφανής.



Σχήμα 5: Το πρόβλημα των πέντε δυαδικών διακοπτών που αναπαριστά την ιδέα της κωδικοποίησης και έναν τρόπο αξιολόγησης.

2. Οι Γενετικοί Αλγόριθμοι κάνουν αναζήτηση σε ένα πλήθος σημείων και όχι σε ένα. Σε πολλές μεθόδους βελτιστοποίησης, μετακινούμαστε προσεχτικά από ένα σημείο του πεδίου ορισμού στο άλλο. Αυτός ο τρόπος προσέγγισης, ενέχει κίνδυνο να περιοριστεί η αναζήτηση σε τοπικό ακρότατο. Οι Γενετικοί Αλγόριθμοι αν δεν εξαλείφουν, τουλάχιστον μειώνουν πάρα πολύ τον κίνδυνο αυτό δουλεύοντας πάνω σε ένα πλήθος σημείων ταυτόχρονα (πλήθος από συμβολοσειρές) ανεβαίνοντας έτσι πολλούς λόφους ταυτόχρονα (hill climbing). Στο παράδειγμα με το μαύρο κουτί και τους διακόπτες, άλλες τεχνικές θα ξεκινούσαν από ένα συνδυασμό των διακοπτών, θα εφάρμοζαν ορισμένους κανόνες μετάβασης και θα παραγόταν ο επόμενος συνδυασμός διακοπτών (ψάξιμο από σημείο σε σημείο). Ένας Γενετικός Αλγόριθμος ξεκινά με ένα πλήθος συμβολοσειρών που αντιπροσωπεύουν αντίστοιχα ένα πλήθος συνδυασμών και κατόπιν παράγει διαδοχικά καινούριους. Για παράδειγμα αν επιλέγαμε μέγεθος πληθυσμού 4 θα μπορούσαμε αρχικά να δημιουργήσουμε έναν τυχαίο πληθυσμό: 01101, 11000, 01000, 10011. Έπειτα τρέχοντας τον αλγόριθμο δημιουργούνται νέοι πληθυσμοί συγκλίνοντας προς το επιθυμητό αποτέλεσμα.
3. Οι Γενετικοί Αλγόριθμοι χρησιμοποιούν πληροφορία που προέρχεται από την αντικειμενική συνάρτηση και όχι άλλες πρόσθετες πληροφορίες. Πολλές μέθοδοι αναζήτησης απαιτούν επιπρόσθετες πληροφορίες για να δουλέψουν σωστά. Οι Γενετικοί Αλγόριθμοι δεν έχουν ανάγκη για τέτοιου είδους πληροφορίες γιατί είναι κατά βάση τυφλοί, δηλαδή αξιοποιούν μόνο την πληροφορία η οποία βρίσκεται στην αντικειμενική συνάρτηση. Αυτό τους προσδίδει μεγάλη ευελιξία, αλλά μερικές φορές οι βοηθητικές

πληροφορίες είναι αναγκαίες για την επίλυση ενός προβλήματος και γι' αυτό το λόγο έχουν αναπτυχθεί μορφές Γενετικών Αλγορίθμων που αξιοποιούν τέτοιες πληροφορίες (Knowledge-Based Genetic Algorithms).

4. Οι Γενετικοί Αλγόριθμοι χρησιμοποιούν πιθανοθεωρητικούς κανόνες μετάβασης και όχι ντετερμινιστικούς. Αυτό όμως δε σημαίνει ότι η αναζήτηση βαδίζει στην τύχη, δηλαδή δεν λαμβάνονται αποφάσεις με το στρίψιμο ενός νομίσματος. Απλά χρησιμοποιείται η τυχαία επιλογή ως ένα εργαλείο για να οδηγηθεί η αναζήτηση σε περιοχές του πεδίου αναζήτησης με πιθανή βελτίωση.

Όλες αυτές οι τέσσερις διαφορές μαζί - άμεση χρήση κωδικοποίησης, αναζήτηση από ένα πληθυσμό, αδιαφορία για επιπρόσθετες πληροφορίες και χρήση τελεστών βασισμένων στην τύχη – συμβάλλουν στην ευρωστία των Γενετικών Αλγορίθμων και τελικά στο πλεονέκτημα που αποκτούν έναντι άλλων κλασσικών μεθόδων που κοινώς χρησιμοποιούνται.

3.3.3. Μειονεκτήματα των Γενετικών Αλγορίθμων

Όπως κάθε μέθοδος, έτσι και οι Γενετικοί Αλγόριθμοι έχουν και αυτοί τα μειονεκτήματά τους. Πολλά μειονεκτήματα πηγάζουν από την δυσπιστία των επιστημόνων για τους Γενετικούς Αλγόριθμους γι' αυτό και παρόλο που οι Γενετικοί Αλγόριθμοι έχουν αναπτυχθεί από τις αρχές της δεκαετίας του εβδομήντα, η εφαρμογή τους άρχισε τα τελευταία χρόνια. Ένας από αυτά τα μειονεκτήματα είναι η εξοικείωση με τη Γενετική [28] όπου για τους περισσότερους που ασχολούνται με την επιστήμη των υπολογιστών, οι έννοιες της Εξέλιξης και της Φυσικής Επιλογής αποτελούν άγνωστο πεδίο και άρα δυσνόητο. Αυτό που συμβαίνει όμως είναι ότι οι Γενετικοί Αλγόριθμοι μιμούνται με αφαιρετικό τρόπο όλες αυτές τις λειτουργίες της φύσης χωρίς να προχωρούν σε μεγάλο επίπεδο λεπτομέρειας. Όπως κάθε μέθοδος έχει το θεωρητικό υπόβαθρό της και τη διατύπωσή της ως προς τον τρόπο λειτουργίας της, έτσι και οι Γενετικοί Αλγόριθμοι έχουν το δικό τους υπόβαθρο που προέρχεται από τη Γενετική. Σε πολλές περιπτώσεις θα ήταν πιο δύσκολη η κατανόηση της διαδικασίας των Γενετικών Αλγορίθμων αν αντί για αναφορά στη Γενετική αναπτύσσονταν απλά οι τεχνικές που χρησιμοποιούν οι Γενετικοί Αλγόριθμοι.

Άλλα μειονεκτήματα τα οποία εντοπίζονται έχουν να κάνουν με το χρόνο. Παρόλο που ένα από τα πλεονεκτήματα που έχουν είναι η καλή εκμετάλλευση της υπολογιστικής ισχύος επειδή χρησιμοποιούν μεγάλο αριθμό πράξεων για την αποτίμηση της συνάρτησης αξιολόγησης [31], ο μεγάλος αριθμός πράξεων προσθέτει μεγάλο υπολογιστικό φόρτο. Ένα άλλο πιο θεωρητικό ερώτημα που τίθεται είναι πώς είναι δυνατό μια μέθοδος αναζήτησης να έχει καλές επιδόσεις χρόνου όταν είναι προερχόμενη από τις φυσικές διαδικασίες οι οποίες εξελίσσονται με ρυθμούς πάρα πολύ αργούς (για να αλλάξουν τα χαρακτηριστικά των ειδών και να διαμορφωθεί η νέα συμπεριφορά τους χρειάζεται να περάσουν χιλιάδες γενιές). Η εξέλιξη όμως εξαρτάται άμεσα από το περιβάλλον οπότε αν οι αλλαγές του περιβάλλοντος γίνονται με ταχύτερο τρόπο, επιταχύνεται και η εξέλιξη. Επιπλέον η κωδικοποίηση σε συμβολοσειρές και η μοντελοποίηση με μαθηματικές σχέσεις δεν παρουσιάζουν ιδιαίτερο υπολογιστικό φόρτο σε σύγκριση με άλλες μεθόδους. Γεγονός αποτελεί ακόμα ότι οι Γενετικοί Αλγόριθμοι δεν εγγυώνται ότι θα δώσουν τη βέλτιστη πάντα λύση καθώς η φύση τους είναι πιθανοθεωρητική, ενώ ένα άλλο γεγονός είναι η ύπαρξη περισσότερο αποδοτικών αλγορίθμων (λόγω εξειδίκευσης) για πολλά συγκεκριμένα προβλήματα και ειδικά για προβλήματα απλά, μικρά, ή μαθηματικής φύσης [32].

3.4. Περιγραφή και Ανάλυση ενός Γενετικού Αλγόριθμου

Στην Ενότητα αυτή γίνεται αναλυτικά η περιγραφή και ανάλυση των βημάτων για την επίλυση ενός προβλήματος μέσω ενός Γενετικού Αλγόριθμου (Σχήμα 6). Οι διαδικασίες που θα περιγραφούν βασίζονται στον Γενετικό Αλγόριθμο τον οποίο αρχικά δημιούργησε ο Holland [29] και είναι γνωστός με την ονομασία Απλός Γενετικός Αλγόριθμος (Simple Genetic Algorithm). Ο αλγόριθμος αυτός εισάγει όλες τις βασικές γενετικές διαδικασίες και αποτελεί μια πολύ καλή πρώτη επαφή για την γνωριμία και επίλυση προβλημάτων με Γενετικούς Αλγόριθμους. Για την περιγραφή των διαδικασιών του Γενετικού

Αλγόριθμου χρησιμοποιείται το παράδειγμα που ανέπτυξε ο Goldberg το 1989 [30] για τη μεγιστοποίηση μιας απλής συνάρτησης.

```
Procedure Genetic Algorithm
begin
    t ← 0
    αρχικοποίησε το P(t)
    αξιολόγησε το P(t)
    while (not συνθήκη τερματισμού) do
    begin
        t ← t +1
        επιλογή του P(t) από το P(t-1)
        τροποποίηση του P(t)
        αξιολόγηση του P(t)
    end
end
```

Σχήμα 6: Η Αλγοριθμική προσέγγιση ενός Γενετικού Αλγόριθμου (όπου με t συμβολίζεται η επαναληπτική εκτέλεση και με P(t) ο πληθυσμός.)

3.4.1. Η Κωδικοποίηση

Η κωδικοποίηση των υποψήφιων λύσεων αποτελεί σημείο αφετηρίας όχι μόνο για τους Γενετικούς Αλγόριθμους αλλά και για κάθε μέθοδο αναζήτησης. Η αναπαράσταση των λύσεων πρέπει να γίνει με ένα μαθηματικά επίσημο τρόπο έτσι ώστε να μπορεί μετέπειτα να μεταφραστεί σε δομές δεδομένων για να δουλέψει στον υπολογιστή. Βασικός στόχος της κωδικοποίησης, είναι να αναπαραστήσει με ικανοποιητικό και αποδοτικό τρόπο τα επιμέρους χαρακτηριστικά των υποψήφιων λύσεων διευκολύνοντας έτσι τις λειτουργίες του αλγόριθμου. Οι περισσότεροι Γενετικοί Αλγόριθμοι χρησιμοποιούν σταθερού μήκους (fixed-length) και ίδιας τάξης (fixed-order) αλφαριθμητικά δυαδικά ψηφία (bit strings), ωστόσο, τελευταία γίνονται πολλά πειράματα με διάφορα άλλα είδη κωδικοποιήσεων [27]. Η επιλογή της δυαδικής κωδικοποίησης χρησιμοποιείται συχνά για διάφορους λόγους, ένας εκ των οποίων είναι ιστορικός: Στα πρώτα στάδια των ερευνών τους, ο Holland και οι συνεργάτες του [27] επικεντρώθηκαν σε τέτοιου είδους κωδικοποιήσεις, οπότε αυτό έγινε κοινή πρακτική των Γενετικών Αλγορίθμων. Από εκεί και έπειτα, υπάρχουν διάφορες επεκτάσεις του βασικού αυτού σχήματος, με κωδικοποιήσεις όπως είναι η GRAY [33] και η diploid binary encoding [29] [30]. Επιπλέον ο Holland [29] έχει δώσει και θεωρητική δικαιολόγηση για την χρήση της δυαδικής κωδικοποίησης. Άλλες κωδικοποιήσεις οι οποίες υπάρχουν, είναι πολλαπλών χαρακτήρων και πραγματικών αριθμών αλλά και δενδρικά σχήματα κωδικοποίησης. Το τελικό συμπέρασμα ωστόσο, είναι ότι οι παράγοντες οι οποίοι καθορίζουν το είδος της κωδικοποίησης που θα πρέπει να επιλέξουμε για ένα πρόβλημα δεν είναι συγκεκριμένοι και εξαρτώνται άμεσα από το είδος του προβλήματος αλλά και από εμπειρικά αποτελέσματα.

Για την κωδικοποίηση του παραδείγματός του Goldberg χρησιμοποιείται η δυαδική αναπαράσταση. Αρχικά η διατύπωση του προβλήματος είναι η εξής: Έστω η συνάρτηση $f(x) = x^2, x \in [0,31]$ και χ ακέραιος. Ζητείται το μέγιστο της συνάρτησης στο πεδίο ορισμού της. Έτσι με τη χρήση πενταψήφιας συμβολοσειράς καλύπτουμε 32 διαφορετικούς δυαδικούς αριθμούς και άρα το πεδίο ορισμού της συνάρτησης που θέλουμε να βελτιστοποιήσουμε. Έτσι, η συμβολοσειρά 01011 αντιστοιχεί στον δεκαδικό αριθμό 18.

3.4.2. Η αντικειμενική συνάρτηση

Το δεύτερο βασικό βήμα για την επίλυση ενός προβλήματος με Γενετικούς Αλγόριθμους είναι ο καθορισμός της συνάρτησης αξιολόγησης ή αντικειμενικής συνάρτησης (evaluation function, objective function). Η αντικειμενική συνάρτηση στους Γενετικούς Αλγόριθμους παίζει το ρόλο του περιβάλλοντος και είναι η μόνη διεπαφή με την πληροφορία. Η αντικειμενική συνάρτηση παίρνει μια αποκωδικοποιημένη συμβολοσειρά και επιστρέφει μια τιμή η οποία υποδεικνύει πόσο καλά λύνει το πρόβλημα η συγκεκριμένη συμβολοσειρά και καθορίζει με αυτό τον τρόπο (μέσω της επιλογής που ακολουθεί) την πιθανότητα επιβίωσης του ατόμου – υποψήφιας λύσης που αντιπροσωπεύει. Για να αποφευχθεί το μειονέκτημα του υπολογιστικού φόρτου, η συνάρτηση αξιολόγησης θα πρέπει να είναι όσο το δυνατόν εύκολα υπολογίσιμη. Στην περίπτωση του παραδείγματός μας, η αντικειμενική συνάρτηση είναι η ίδια η συνάρτηση που θέλουμε να μεγιστοποιήσουμε.

3.4.3. Γενετικές διαδικασίες και τελεστές

Μετά και από το δεύτερο βήμα για την επίλυση ενός προβλήματος με Γενετικούς Αλγόριθμους, ακολουθεί το ανεξάρτητο κομμάτι του Γενετικού Αλγόριθμου, οι γενετικές διαδικασίες. Οι γενετικές διαδικασίες δεν εξαρτώνται από ποια κωδικοποίηση ή αναπαράσταση θα χρησιμοποιήσουμε. Έτσι, κάποιος θα μπορούσε να αλλάζει κωδικοποίηση ή αναπαράσταση ή και τα δύο, χωρίς να αναγκαστεί να αλλάζει άλλο σημείο στον κώδικα. Οι λειτουργίες που περιλαμβάνουν τις γενετικές διαδικασίες αποτελούν και τον κύριο φόρτο που παράγεται από τον αλγόριθμο και η τυπική δομή τους είναι η παρακάτω:

1. Αρχικοποίηση (Initialization)
2. Αποκωδικοποίηση (Decoding)
3. Υπολογισμός αντικειμενικής συνάρτησης (Fitness Evaluation)
4. Αναπαραγωγή (Reproduction)
    - Επιλογή (Selection)
    - Διασταύρωση (Crossover)
    - Μετάλλαξη (Mutation)
5. Επανάληψη από το 2^{ο} βήμα μέχρι ικανοποίησης του κριτηρίου τερματισμού του Γενετικού Αλγόριθμου.

Η αρχικοποίηση που γίνεται στο πρώτο βήμα, περιλαμβάνει τη δημιουργία του αρχικού πληθυσμού η οποία τις περισσότερες φορές γίνεται με τυχαίο τρόπο ανάμεσα στις δυνατές τιμές που μπορούν να πάρουν οι μεταβλητές – υποψήφιες λύσεις στο πρόβλημα. Σε άλλες περιπτώσεις η δημιουργία του αρχικού πληθυσμού γίνεται με ευρετικό τρόπο. Στο παράδειγμά μας έχει επιλεχτεί ο πληθυσμός να αποτελείται από τέσσερα άτομα τα οποία μπορεί να προκύψουν με 20 διαδοχικές ρίψεις ενός αμερόληπτου νομίσματος (4 συμβολοσειρές μήκους 5 η καθεμία). Έτσι με τον τρόπο αυτό υποθέτομε ότι προκύπτουν οι παρακάτω συμβολοσειρές: 01101, 11000, 01000 και 10011 **Error! Reference source not found.**(Πίνακας 2). Στο επόμενο βήμα αρχίζει η επαναληπτική διαδικασία του αλγόριθμου. Για να μπορέσει να αξιολογηθεί ο πληθυσμός με την συνάρτηση αξιολόγησης θα πρέπει να γίνει η αποκωδικοποίηση των ατόμων. Όπως τα χαρακτηριστικά των ατόμων στη φύση είναι κωδικοποιημένα στο γονότυπο και η ορατή εμφάνιση αυτών στο περιβάλλον αποτελεί το φαινότυπο, έτσι και στους Γενετικούς Αλγόριθμους τα δυαδικά ψηφία παίζουν το ρόλο του γονότυπου, ενώ στο τεχνητό περιβάλλον, δηλαδή την αντικειμενική συνάρτηση, εμφανίζεται ο φαινότυπος που αντιστοιχεί στη αποκωδικοποιημένη συμβολοσειρά. Αφού αξιολογηθούν οι τιμές για κάθε άτομο στον πληθυσμό εφαρμόζονται οι γενετικοί τελεστές της αναπαραγωγής και η επαναξιολόγηση μέχρι να φτάσει ο αλγόριθμος σε συνθήκη τερματισμού.

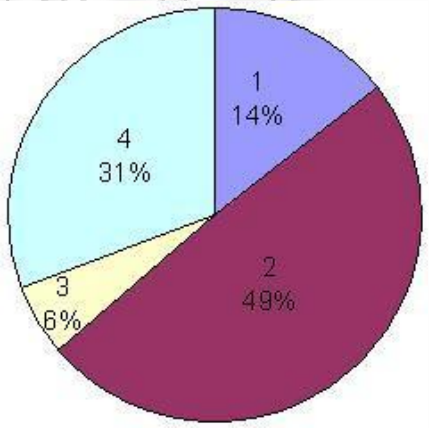
Αριθμός Συμβολοσειράς	Συμβολοσειρά (κωδικοποιημένη)	Απόδοση	Απόδοση %
1	01101	169	14.4

2	11000	576	49.5
3	01000	64	5.5
4	10011	361	30.9
Σύνολο		1170	100.0

Πίνακας 2: Αρχικοποίηση πληθυσμού και αξιολόγηση

3.4.4. Η επιλογή

Η πρώτη απόφαση, στα πλαίσια του καθορισμού των γενετικών διαδικασιών, είναι η επιλογή. Δηλαδή πώς θα επιλεχτούν τα άτομα από τον πληθυσμό τα οποία θα δημιουργήσουν τους απογόνους για την επόμενη γενιά. Ο σκοπός της επιλογής, είναι η αύξηση των ικανότερων ατόμων του πληθυσμού ώστε να επικρατήσουν σε επόμενες γενιές και οι απόγονοί τους να είναι ακόμα ικανότεροι αφού συνδυάζουν τα καλύτερα χαρακτηριστικά των γονέων τους. Ένας Γενετικός Αλγόριθμος χωρίς την λειτουργία της επιλογής είναι στην ουσία ένας αλγόριθμος τυχαίας αναζήτησης (brute force). Όπως και στην κωδικοποίηση, έτσι και στη επιλογή έχουν προταθεί αρκετά σχήματα τα οποία εξυπηρετούν την επιλογή. Ο τρόπος που χρησιμοποιήθηκε από την αρχή από τον Holland είναι αυτός της αναλογικής επιλογής (proportionate selection) στον οποίο η αναμενόμενη τιμή ενός ατόμου (expected value) είναι η τιμή που του δόθηκε από την αντικειμενική συνάρτηση διαιρούμενη με το μέσο όρο της τιμής της αντικειμενικής συνάρτησης όλου του πληθυσμού. Η πιο συνήθης μέθοδος για την υλοποίηση της παραπάνω επιλογής είναι η εξαναγκασμένη ρουλέτα (roulette wheel selection). Στην ρουλέτα αυτή, για κάθε άτομο του πληθυσμού, αντιστοιχούμε μερίδιο τόσο, όσο και η τιμή της αντικειμενικής του συνάρτησης αναλογικά πάντα με ολόκληρο τον πληθυσμό. Για την επιλογή, στρίβουμε τη ρουλέτα τόσες φορές, όσες είναι και τα άτομα στον πληθυσμό. Με τον τρόπο αυτόν αναμένουμε στατιστικά ότι τα άτομα του πληθυσμού με μεγαλύτερη τιμή αξιολόγησης (άρα και μεγαλύτερη επιφάνεια στη ρουλέτα) θα εμφανίζουν μεγαλύτερο αριθμό επιλογών από τα υπόλοιπα.



Σχήμα 7: Σχηματική αναπαράσταση εξαναγκασμένης ρουλέτας

Στο παράδειγμά μας στρίβουμε τη ρουλέτα τέσσερις φορές. Παρατηρούμε ότι ανάλογα και με την απόδοση της κάθε συμβολοσειράς (Πίνακας 2), αντίστοιχα κατανέμεται και η επιφάνεια στη ρουλέτα (Σχήμα 7). Ο νέος πληθυσμός που σχηματίζεται από την επιλογή ονομάζεται mating pool [30] (δεξαμενή ζευγαρώματος).

Άλλη εναλλακτική μέθοδος επιλογής είναι και η επιλογή ταξινόμησης (rank selection) [34] η οποία είναι μια από τις μεθόδους που υλοποιήθηκαν ώστε να αποφύγουν το μειονέκτημα της πρόωρης σύγκλισης (premature convergence) στην αναλογική επιλογή: Στα πρώτα στάδια του αλγορίθμου, τα άτομα του πληθυσμού έχουν μεγάλη διακύμανση στις τιμές αξιολόγησης ενώ μερικά από αυτά έχουν αρκετά μεγαλύτερη τιμή αξιολόγησης. Στην αναλογική επιλογή, τα άτομα αυτά και οι απόγονοί τους θα πολλαπλασιάζονται γρήγορα μέσα στον πληθυσμό εμποδίζοντας τον αλγόριθμο να κάνει επιπλέον

εξερεύνηση του χώρου αναζήτησης. Επιπλέον, μετά από αρκετές γενιές όπου η διακύμανση στις τιμές αξιολόγησης των ατόμων θα είναι μικρές, δεν θα υπάρχουν ουσιαστικές διαφορές στην αξιολόγηση για να τις εκμεταλλευθεί η επιλογή οπότε και η εξέλιξη σταματά. Στην επιλογή ταξινόμησης [34], τα άτομα στον πληθυσμό κατατάσσονται ανάλογα με την τιμή αξιολόγησής τους και η αναμενόμενη τιμή τους εξαρτάται από τη θέση που βρίσκονται στην κατάταξη και όχι απόλυτα από την τιμή της αξιολόγησής τους.

Στην εξαναγκασμένη ρουλέτα πρέπει να γίνουν δύο σαρώσεις στον πληθυσμό: μία για να υπολογιστεί η μέση τιμή αξιολόγησης και άλλη μια σάρωση για να υπολογιστεί η αναμενόμενη τιμή κάθε ατόμου. Στην rank επιλογή πρέπει να γίνει ταξινόμηση όλου του πληθυσμού, διαδικασία που ειδικά σε μεγάλο αριθμό πληθυσμού καταναλώνει υψηλό υπολογιστικό χρόνο. Η επιλογή τουρνουά (tournament selection) [35], παρέχει τη φιλοσοφία της rank επιλογής με τη διαφορά ότι είναι πιο αποδοτική υπολογιστικά και ενδείκνυται για παράλληλη υλοποίηση. Η επιλογή τουρνουά μπορεί να είναι δυαδική ή N-αδική. Ανάλογα με την υλοποίηση, εκλέγονται τυχαία, δύο ή N άτομα από τον πληθυσμό και το καλύτερο από αυτά επιλέγεται ως γονέας. Έπειτα, τα άτομα που εκλέχθηκαν, ξαναμπαίνουν στη λίστα και ξαναρχίζει η διαδικασία.

Στο παράδειγμα του χ^2, η διαδικασία της επιλογής φαίνεται στον παρακάτω πίνακα (Πίνακας 3).

Αριθμός Συμβολοσειράς	Αρχικός πληθυσμός	Τιμή του χ	Αξιολόγηση $f(\chi) = \chi^2$	Πιθανότητα επιλογής $pselect_i = \frac{f_i}{\sum f}$	Αναμενόμενος αριθμός αντιγράφων $\frac{f_i}{\overline{f}}$	Αριθμός αντιγράφων από τη ρουλέτα
1	01101	13	169	0.14	0.58	1
2	11000	24	576	0.49	1.97	2
3	01000	8	64	0.06	0.22	0
4	10011	19	361	0.31	1.23	1
Σύνολο $\sum f$			1170	1.00	4.00	4
Μέσος όρος $\sum f/4$			293	0.25	1.00	1
Μέγιστο			576	0.49	1.97	2

Πίνακας 3: Διαδικασία επιλογής από πληθυσμό

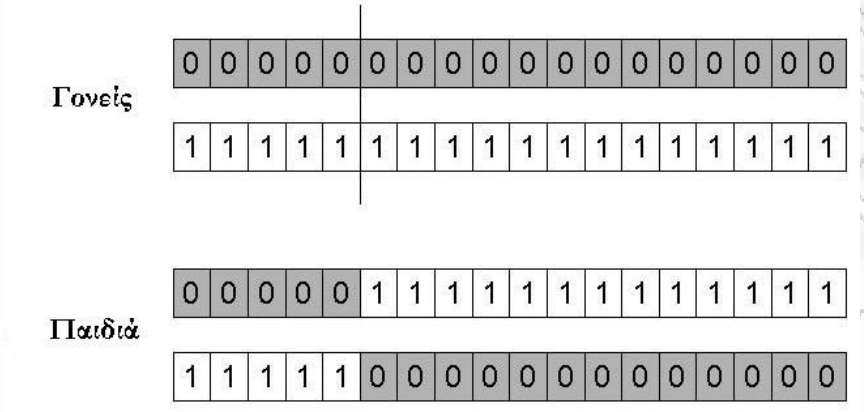
Στον παραπάνω πίνακα βλέπουμε πως η αξιολόγηση (στήλη 4) των πιθανών λύσεων (στήλη 2) μεταφράζεται σε πιθανότητα επιλογής (στήλη 5) και στον αριθμό των αντιγράφων που θα προκύψει από την εφαρμογή της επιλογής μέσω της εξαναγκασμένης ρουλέτας (στήλη 7).

3.4.5. Η Διασταύρωση

Από τη διαδικασία της επιλογής έχει προκύψει ένας προσωρινός πληθυσμός ο οποίος είναι αυτός που θα ζευγαρωθεί ώστε να γίνει μια παρόμοια διαδικασία με αυτήν της φύσης. Η διασταύρωση στους Γενετικούς Αλγόριθμους αποτελεί μια σημαντική διαδικασία και συμβάλλει αποφασιστικά στην επίδοση και την επιτυχία του. Η διασταύρωση εξυπηρετεί την ανταλλαγή πληροφοριών μεταξύ διαφορετικών πιθανών λύσεων και σκοπός της είναι η νέα γενιά, που θα προκύψει αμέσως μετά, να περιέχει άτομα τα οποία φέρουν τα καλύτερα χαρακτηριστικά των γονέων τους. Επίσης, η διασταύρωση έχει την ιδιότητα να ανακατευθύνει την αναζήτηση σε νέες περιοχές του χώρου αναζήτησης. Η διασταύρωση βέβαια δεν δίνει πάντα καλύτερα αποτελέσματα αλλά τα χειρότερα παιδιά που τυχόν δημιουργηθούν στη νέα γενιά, θα έχουν μικρότερη πιθανότητα να επιλεγούν για την επόμενη.

Η διασταύρωση λαμβάνει χώρα ως εξής: Επιλέγονται τυχαία από τον νέο πληθυσμό που έχει προκύψει, ομάδες των δύο ατόμων. Μεταξύ ποιων ατόμων θα γίνει η διασταύρωση αποτελεί

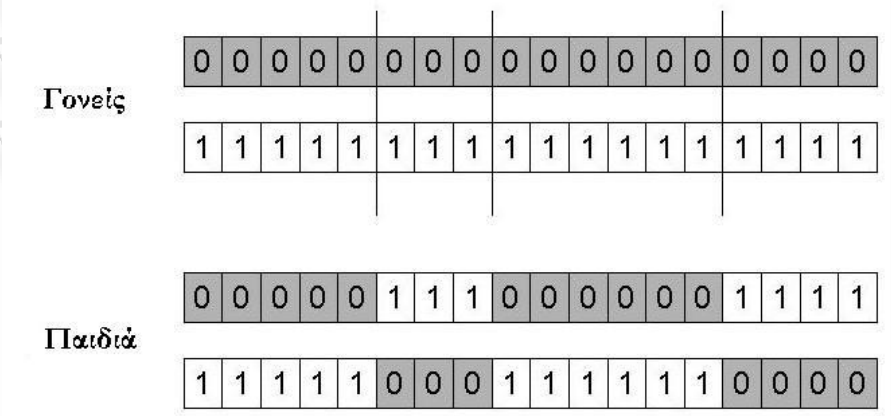
αντικείμενο μελέτης και μέχρι τώρα στη βιβλιογραφία το ζευγάρωμα γίνεται με τυχαίο τρόπο. Επιλέγεται επίσης ένα σημείο διασταύρωσης τυχαία. Χωρίζονται οι γονείς σε αυτό το σημείο και δημιουργούνται τα παιδιά ανταλλάσσοντας τα τμήματα στα οποία χωρίστηκαν οι γονείς (Σχήμα 8).



Σχήμα 8: Σχηματική αναπαράσταση διασταύρωσης

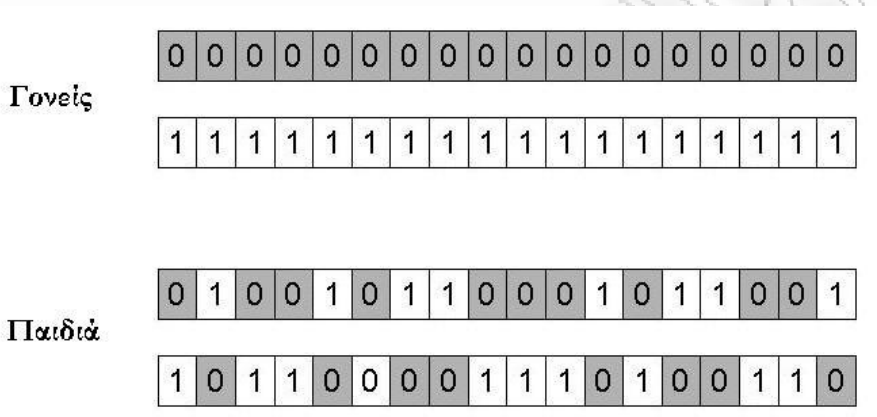
Η διασταύρωση λαμβάνει μέρος παραμετρικά, δηλαδή συμβαίνει με μια συγκεκριμένη πιθανότητα, την πιθανότητα διασταύρωσης (crossover probability) και συμβολίζεται ως p_c. Η μεταβλητή αυτή καθορίζεται από τον σχεδιαστή του αλγορίθμου και μπορεί να ποικίλει από πρόβλημα σε πρόβλημα και από τρέξιμο σε τρέξιμο του ίδιου αλγόριθμου. Η τιμή της πιθανότητας αυτής επηρεάζει την σύγκλιση του αλγόριθμου προς πιθανές λύσεις και κατ' επέκταση το χρόνο τρεξίματος του αλγόριθμου. Αν η διασταύρωση είναι απόλυτη, δηλαδή αν p_c=1, τότε όλα τα άτομα διασταυρώνονται με αποτέλεσμα η αναζήτηση να επεκτείνεται σε όλο το χώρο, οπότε ο αλγόριθμος θα συγκλίνει προς το βέλτιστο με αργό ρυθμό. Απ' την άλλη, αν η διασταύρωση γίνεται με μικρή πιθανότητα, το ψάξιμο στον χώρο αναζήτησης γίνεται με μεγάλα βήματα συγκλίνοντας πιο γρήγορα αλλά με κίνδυνο να προσπεράσει τη βέλτιστη.

Η διασταύρωση ενός σημείου που περιγράφηκε είναι η απλούστερη μορφή. Έχει μερικές αδυναμίες όπως η πιθανότητα να διατηρεί γονότυπους οι οποίοι μοιάζουν μεταξύ τους και η αδυναμία να κρατήσει γονίδια από δύο αντίθετα μέρη του γονότυπου, αδυναμία που έχει ονομαστεί positional bias. Αυτές οι αδυναμίες φαίνονται πιο ξεκάθαρα με την θεωρία σχημάτων όπου τεκμηριώνονται πιο θεωρητικά και με μαθηματικό τρόπο η απόδοση των Γενετικών Αλγορίθμων. Για την μείωση τέτοιων αδυναμιών αλλά και από ερευνητικό ενδιαφέρον, έχουν αναπτυχθεί διάφορες άλλες τεχνικές που εξυπηρετούν τη διασταύρωση. Μια τεχνική που ουσιαστικά είναι η γενίκευση της απλής διασταύρωσης, είναι η διασταύρωση n σημείων (n-point crossover), όπου επιλέγουμε τυχαία n σημεία διασταύρωσης και αφού χωρίσουμε τα κομμάτια, τα συνενώνουμε εναλλάξ μεταξύ των γονέων (Σχήμα 9). Και στην περίπτωση αυτή, θεωρούμε ότι έχουμε κι εδώ μια πιο ήπιας μορφής positional bias.



Σχήμα 9: Σχηματική αναπαράσταση διασταύρωσης n-σημείων

Άλλη μια εναλλακτική διασταύρωση είναι η ομοιόμορφη (uniform crossover). Εδώ αναθέτουμε «κορώνα» στον ένα γονέα και «γράμματα» στον άλλον και για να δημιουργήσουμε το πρώτο παιδί στρίβουμε ένα μη προκατειλημμένο νόμισμα για κάθε γονίδιο του παιδιού. Αν έρθει κορώνα το παιδί παίρνει το γονίδιο από τον πρώτο γονέα, ενώ αν έρθει γράμματα παίρνει από τον δεύτερο. Η δημιουργία του δεύτερου παιδιού γίνεται απλώς από τα γονίδια που δεν έχουν επιλεγεί από την δημιουργία του πρώτου (Σχήμα 10).



Σχήμα 10: Σχηματική αναπαράσταση της ομοιόμορφης διασταύρωσης

Έκτός από τις βασικές αυτές μορφές διασταύρωσης υπάρχει πλήθος άλλων διασταυρώσεων αλλά και άλλες οι οποίες εφαρμόζονται σε συγκεκριμένες κωδικοποιήσεις όπως ακεραίων και πραγματικών αριθμών. Στο παράδειγμα του χ^2, η διαδικασία της διασταύρωσης φαίνεται μέσα από τον παρακάτω πίνακα (Πίνακας 4).

Αριθμός Συμβολοσειράς	Πληθυσμός μετά τη επιλογή (mating pool)	Ζευγάρωμα (τυχαία επιλεγμένο)	Σημείο διασταύρωσης (τυχαία επιλεγμένο)	Νέος πληθυσμός	Τιμή του χ	Αξιολόγηση $f(\chi) = \chi^2$
1	0110|1	2	4	01100	12	144
2	1100|0	1	4	11001	25	625
3	11|000	4	2	11011	27	729
4	10|011	3	2	10000	16	256
Σύνολο $\sum f$						1754
Μέσος όρος $\sum f/4$						439
Μέγιστο						729

Πίνακας 4: Διαδικασία διασταύρωσης

Ήδη παρατηρούμε ότι αμέσως μετά την επιλογή και διασταύρωση του νέου πληθυσμού, η αξιολόγηση της αντικειμενικής συνάρτησης δείχνει αύξηση τόσο στη μέγιστη πιθανή λύση (729 έναντι

576 της αρχικής) όσο και στο μέσο όρο (439 έναντι 293). Αμέσως μετά από τη διασταύρωση ακολουθεί και η μετάλλαξη και τότε προκύπτει πλήρως η νέα γενιά.

3.4.6. Η μετάλλαξη

Από την αρχή της δημιουργίας των Γενετικών Αλγορίθμων και αναφερόμενοι πίσω στον Holland και το βιβλίο του *Adaptation in Natural and Artificial Systems*, [29] η διασταύρωση κατείχε τον κύριο λόγο στους γενετικούς τελεστές. Η μετάλλαξη έπαιζε πιο πολύ υποστηρικτικό ρόλο με την έννοια ότι εξασφάλιζε τον πληθυσμό από πλήρη σταθεροποίηση σε συγκεκριμένες θέσεις στα άτομά του. Η εκτίμηση ωστόσο τα τελευταία χρόνια για την μετάλλαξη έχει αυξηθεί καθώς γίνεται προσπάθεια για επίλυση όλο και πιο πολύπλοκων προβλημάτων με Γενετικούς Αλγόριθμους. Διάφορες έρευνες έχουν γίνει για να αναδείξουν ότι η μετάλλαξη έχει ισχυρότερο ρόλο από αυτόν που έχει στις κλασσικές διατάξεις των Γενετικών Αλγορίθμων [36] ή για να συγκρίνουν τους δύο αυτούς τελεστές [37] αλλά αυτό που τελικά μας δείχνουν τα πράγματα, δεν είναι η επιλογή ανάμεσα στην διασταύρωση ή την μετάλλαξη, αλλά η ισορροπία μεταξύ των δύο αυτών παραμέτρων αλλά και της επιλογής. Η ισορροπία αυτή εξαρτάται με τη σειρά της από άλλους παράγοντες όπως η αντικειμενική συνάρτηση και η κωδικοποίηση. Επιπλέον, το πόσο χρήσιμη είναι η διασταύρωση και η μετάλλαξη σε κάθε γενιά του Γενετικού Αλγόριθμου, δεν είναι ξεκαθαρισμένο ακόμα.

Η λειτουργία της μετάλλαξης είναι να αλλάζει αυθαίρετα ένα ή περισσότερα γονίδια ενός συγκεκριμένου χρωμοσώματος – ατόμου και λαμβάνει μέρος, όπως και η διασταύρωση, παραμετρικά με μια πιθανότητα (mutation probability) που συμβολίζουμε p_m. Η μετάλλαξη εξυπηρετεί την εισαγωγή νέων πιθανών λύσεων στον ήδη υπάρχοντα πληθυσμό, ενώ παράλληλα ανακατευθύνει την αναζήτηση και την επεκτείνει σε όλο το χώρο αναζήτησης. Ο τρόπος με τον οποίο δρα, είναι η αντιγραφή των γονιδίων από τον γονέα στον απόγονο και ανάλογα με την πιθανότητα μετάλλαξης, αντιστρέφει ένα γονίδιο π.χ. από 0 σε 1 και το αντίστροφο. Η πιθανότητα μετάλλαξης πρέπει οπωσδήποτε να είναι χαμηλή (της τάξης του 1 στα χίλια) γιατί σε διαφορετική περίπτωση ο αλγόριθμος καταλήγει σε τυχαίο ψάξιμο. Στο παράδειγμα του χ^2, η διαδικασία της μετάλλαξης, η οποία εφαρμόζεται στον καινούριο πληθυσμό που προέκυψε μετά την επιλογή και διασταύρωση, φαίνεται στον Πίνακας 5.

Αριθμός Συμβολοσειράς	Πληθυσμός μετά την διασταύρωση	Πληθυσμός μετά την μετάλλαξη	Τιμή του χ	Αξιολόγηση $f(\chi) = \chi^2$
1	01100	**1**1100	26	676
2	11001	11001	25	625
3	11011	11011	27	729
4	10000	10**1**00	18	324
Σύνολο $\sum f$				2354
Μέσος όρος $\sum f/4$				588.5
Μέγιστο				729

Πίνακας 5: Διαδικασία μετάλλαξης

Και σε αυτή την περίπτωση παρατηρούμε την εισαγωγή δύο νέων λύσεων οι οποίες παρόλο που δεν αυξάνουν τη μέγιστη πιθανή λύση, ανεβάζουν το μέσο όρο αξιολόγησης, δηλαδή αυξάνουν την ικανότητα των ατόμων μέσα στον πληθυσμό. Σε πολλές περιπτώσεις βέβαια, η μετάλλαξη θα μπορούσε να δώσει χειρότερες νέες λύσεις, αλλά όπως και στη διασταύρωση, οι νέες αυτές λύσεις θα έχουν μικρότερη πιθανότητα επιλογής στην επόμενη γενιά.

3.5. Διάφορα θέματα υλοποίησης

3.5.1. Χειρισμός Περιορισμών

Τις περισσότερες φορές που έχουμε μια συνάρτηση να βελτιστοποιήσουμε ή ένα πρόβλημα να μοντελοποιήσουμε με γενετικούς αλγόριθμους, προκύπτουν περιορισμοί οι οποίοι πρέπει να ικανοποιούνται. Οι περιορισμοί στις αντικειμενικές συναρτήσεις εμφανίζονται με την μορφή ανισοτήτων ενώ για το χειρισμό τους υπάρχουν διάφοροι τρόποι. Στην απλούστερη περίπτωση ο αλγόριθμος αγνοεί τις λύσεις εκτός του πεδίου ορισμού, καταχωρώντας του μηδενική απόδοση τις περισσότερες φορές όμως θέλουμε να εκμεταλλευτούμε την πληροφορία που έχουν οι λύσεις αυτές. Ένας τρόπος για να ενσωματώσουμε τους περιορισμούς και να εκμεταλλευτούμε την πληροφορία, είναι η μέθοδος της ποινής (penalty method), η οποία χειρίζεται τις παραβιάσεις των περιορισμών δίνοντας ανάλογα ένα κόστος (penalty). Η μορφή της τεχνικής αυτής είναι ως εξής:

$$eval(x) = \begin{cases} f(x), \text{if } x \text{ is feasible} \\ f(x) - penalty(x), \text{otherwise,} \end{cases}$$

Όπου penalty(*x*) είναι 0, αν δεν συμβαίνει παραβίαση των περιορισμών και θετικό αν συμβαίνει παραβίαση. Θα πρέπει να σημειωθεί ωστόσο, ότι μια μέθοδος ποινής σε σχέση με μια άλλη, κατέχουν ειδοποιείς διαφορές στον τρόπο που σχεδιάζονται και εφαρμόζονται στις μη εφικτές λύσεις.

3.5.2. Επιλογή παραμέτρων

Τελευταία αλλά εξίσου σημαντική επιλογή στην υλοποίηση ενός γενετικού αλγόριθμου, είναι η επιλογή των διαφόρων παραμέτρων, όπως το μέγεθος του πληθυσμού χρωμοσωμάτων, η πιθανότητα διασταύρωσης και η πιθανότητα μετάλλαξης. Η επιλογή αυτών των παραμέτρων είναι δύσκολη απόφαση εξαιτίας των ποικίλων παραλλαγών που μπορεί να έχει ένας γενετικός αλγόριθμος καθώς και της διαφορετικής κάθε φορά συνάρτησης αξιολόγησης. Εξάλλου, ένας γενετικός αλγόριθμος στηρίζεται στην τύχη κάθε φορά που εκτελεί δημιουργία πληθυσμού, επιλογή, διασταύρωση ή μετάλλαξη. Έτσι σε κάθε επανάληψή του, με διαφορετικό παράγοντα τύχης (random number seed), παράγει και διαφορετικά αποτελέσματα. Επιπλέον στα παραπάνω, προστίθεται και η ύπαρξη διαφόρων τύπων διασταύρωσης, μετάλλαξης και διαφορετικών τρόπων κωδικοποίησης. Η σύγκριση όλων αυτών των διαφορετικών επιλογών και ο υπολογισμός του μέσου όρου των αποτελεσμάτων, ώστε να μειωθούν οι τυχαίες αποκλίσεις, καθιστούν τη διαδικασία αρκετά επίπονη [38].

Η πρώτη συστηματική μελέτη των παραμέτρων των γενετικών αλγορίθμων, έγινε από τον De Jong (1975) στα πλαίσια της διδακτορικής του διατριβής [39] στην οποία μετέτρεψε τις θεωρίες του Holland στην πράξη, σε βελτιστοποίηση συναρτήσεων. Στο βιβλίο του ο Goldberg [30] συνοψίζει τα αποτελέσματα αυτά. Δύο βασικά κριτήρια απόδοσης που εισήλθαν στην έρευνα αυτή είναι η άμεσα συνδεδεμένα απόδοση (on-line performance) η οποία ισούται με το μέσο όρο από όλα τα κόστη της συνάρτησης αξιολόγησης μέχρι την τρέχουσα γενιά, και έμμεσα συνδεδεμένη απόδοση (off-line performance) η οποία ισούται με το μέσο όρο από τα κόστη της τρέχουσας γενιάς. Το πρώτο κριτήριο «τιμωρεί» τον αλγόριθμο για πολλές φτωχές υποψήφιες λύσεις και τον «ανταμείβει» για το πόσο γρήγορα συγκλίνει προς τις καλύτερες. Αυτά που κατέληξε ο De Jong τελικά, ήταν:

1. Ο μικρού μεγέθους πληθυσμός βελτίωσε την αρχική απόδοση, ενώ ο μεγάλου μεγέθους βελτίωσε τη μακροπρόθεσμη απόδοση. Το καλύτερο μέγεθος πληθυσμού κυμάνθηκε: $50 \leq N_{pop} \leq 100$.
2. Υψηλός ρυθμός μετάλλαξης ήταν καλός στην έμμεσα συνδεδεμένη απόδοση, ενώ χαμηλός ρυθμός μετάλλαξης ήταν καλός στην άμεσα συνδεδεμένη απόδοση. Ο καλύτερος ρυθμός ήταν: $\mu = 0{,}001$.
3. Ο καλύτερος ρυθμός διασταύρωσης ήταν $X_{rate} \approx 0{,}6$.

4. Ο τύπος της διασταύρωσης (ενός σημείου έναντι πολλών σημείων) είχε μικρή διαφορά.

Μια δεκαετία αργότερα, έγινε άλλη μια σημαντική έρευνα πάνω στο θέμα των παραμέτρων ενός γενετικού αλγόριθμου από τον Grefenstette (1986) [40]. Στην έρευνα αυτή χρησιμοποιήθηκε ένας μεταγενετικός αλγόριθμος για να βελτιστοποιήσει τις παραμέτρους για τους υπόλοιπους. Ο αλγόριθμος αυτός χρησιμοποίησε τις μέχρι τότε προτεινόμενες παραμέτρους δηλαδή $N_{pop} = 50$, $X_{rate} = 0,6$, και $\mu = 0,001$. Το καλύτερο αποτέλεσμα ήταν: $N_{pop} = 30$, $X_{rate} = 0,95$, και $\mu = 0,01$.

Άλλες έρευνες, όπως αυτή των Schaffer, Caruana, Eshelman, και Das (1989) έδειξαν παρόμοια αποτελέσματα: $20 \leq N_{pop} \leq 30$, $0,75 \leq X_{rate} \leq 0,95$, και $0,005 \leq \mu \leq 0,01$.

Τέλος, μια άλλη προσέγγιση (Eiben - 1999) [41] είναι οι μεταβλητές παράμετροι και ο έλεγχός τους μέσω άλλης παραμέτρου της οποίας η διαβάθμιση περιλαμβάνει:

- Ντετερμινιστική παράμετρο ελέγχου (deterministic parameter control). Οι παράμετροι του γενετικού αλγόριθμου αλλάζουν σύμφωνα με έναν ντετερμινιστικό κανόνα.
- Προσαρμόσιμη παράμετρο ελέγχου (adaptive parameter control). Οι παράμετροι του γενετικού αλγόριθμου αλλάζουν σύμφωνα με μια ανάδραση που εξασφαλίζει ο αλγόριθμος.
- Αυτό-προσαρμόσιμη παράμετρο ελέγχου (self-adaptive parameter control). Οι παράμετροι του γενετικού αλγόριθμου είναι κωδικοποιημένες μέσα στα χρωμοσώματα και βελτιστοποιούνται ταυτόχρονα με την συνάρτηση αξιολόγησης.

3.5.3. Παράλληλοι γενετικοί αλγόριθμοι

Οι παράλληλοι γενετικοί αλγόριθμοι πολλές φορές έχουν να ασχοληθούν με πολύπλοκες συναρτήσεις αξιολόγησης, ειδικά τα τελευταία χρόνια όπου συνεχώς προκύπτουν πιο περίπλοκα προβλήματα. Ο υπολογισμός των συναρτήσεων αυτών απαιτεί μεγάλη υπολογιστική ισχύ, καθώς ένας γενετικός αλγόριθμος κάνει πολλούς τέτοιους υπολογισμούς κατά τη διάρκεια της βελτιστοποίησης. Ένα θετικό χαρακτηριστικό όμως είναι η δυνατότητά τους για παράλληλη υλοποίηση. Επειδή οι υπολογισμοί των συναρτήσεων αξιολόγησης είναι ανεξάρτητοι μεταξύ τους, μπορούν να γίνονται παράλληλα με μικρές αλλαγές στον κώδικα. Ωστόσο, η παράλληλη υλοποίηση σε συναρτήσεις αξιολόγησης με μέτρια ή χαμηλή απαίτηση υπολογιστικής ισχύος, καταναλώνει το κέρδος από την παράλληλη υλοποίηση στην επικοινωνία μεταξύ των επεξεργαστών [38]. Αυτό συμβαίνει κυρίως επειδή ένας συμβατικός γενετικός αλγόριθμος, πραγματοποιεί την επιλογή του πληθυσμού αφότου έχει δημιουργηθεί ο καινούριος πληθυσμός. Για το λόγο αυτό, χρειάζεται μεγάλη προσοχή στους χειρισμούς που θα κάνουμε για την παράλληλη υλοποίηση.

Το σημαντικότερο πλεονέκτημα της παράλληλης υλοποίησης είναι η επιτάχυνση της διαδικασίας. Ωστόσο, η ταχύτητα δεν είναι το μόνο κίνητρο για την επίλυση της παράλληλης υλοποίησης. Υπάρχουν στρατηγικές υλοποίηση παραλληλισμού, όπου ομάδες των χρωμοσωμάτων στέλνονται σε διαφορετικό επεξεργαστή για να εξελιχθούν ξεχωριστά από τα υπόλοιπα, ενώ η επικοινωνία γίνεται περιστασιακά. Ο διαχωρισμός αυτός σε υπο-πληθυσμούς, μπορεί να αποτρέψει τον αλγόριθμο από πιθανή πρόωρη σύγκλιση, γιατί κάθε πληθυσμός ψάχνει με διαφορετικούς συνδυασμούς αποτρέποντας με τη σειρά του ένα και μόνο υψηλής αξιολόγησης χρωμόσωμα να επικρατήσει σε όλο τον πληθυσμό. Άλλη ένα χαρακτηριστικό της εξέλιξης το οποίο μπορεί να υιοθετήσει η παράλληλη υλοποίηση, είναι η περιστασιακή μετανάστευση μεταξύ των πληθυσμών όπως συναντάται και στη φύση.

Όσον αφορά στην μέθοδο που μπορούμε να ακολουθήσομε για παράλληλη υλοποίηση, η απλούστερη είναι master-slave προσέγγιση. Ένας επεξεργαστής παίζει το ρόλο του master και είναι υπεύθυνος για την επικοινωνία, την ταξινόμηση και το ζευγάρωμα του πληθυσμού ενώ στέλνει τη συνάρτηση αξιολόγησης προς τους slave επεξεργαστές για παράλληλες αποτιμήσεις. Παρόλη την απλότητα ως προς την υλοποίηση, η μέθοδος αυτή έχει ορισμένα μειονεκτήματα: Ο master επεξεργαστής πολλές φορές αναγκάζεται σε αναμονή των αποτελεσμάτων από τους slave, ενώ ο αλγόριθμος είναι τόσο γρήγορος, όσο γρήγορη είναι η επικοινωνία με τον αργότερο υπο-πληθσμό.

4. Προσέγγιση των Γενετικών Αλγορίθμων στην ανίχνευση εισβολών

Σε αυτή την ενότητα αποτυπώνεται αφενός η έρευνα που έχει γίνει μέχρι τώρα στην ανίχνευση και εξόρυξη δικτυακών εισβολών με χρήση Γενετικών Αλγορίθμων και αφετέρου αναλύεται η προτεινόμενη προσέγγιση στην παρούσα υλοποίηση.

4.1. Σχετική Έρευνα

Την τελευταία δεκαετία πολλές τεχνικές προερχόμενες από την τεχνητή νοημοσύνη και πιο συγκεκριμένα τις λεγόμενες «Soft Computing» τεχνικές (Γενετικοί Αλγόριθμοι, Γενετικός Προγραμματισμός, Νευρωνικά Δίκτυα, Ασαφή Συστήματα, χαοτικά συστήματα κ.α.) έχουν επιστρατευθεί για την αποδοτική και αξιόπιστη ανίχνευση εισβολών.

Μια πρώτη προσέγγιση ευφυών τεχνικών στο πρόβλημα ήταν των Crosbie και Spafford [42], όπου με την εφαρμογή Γενετικού Προγραμματισμού (GP) και πολύ-πρακτορικών τεχνικών επιχειρήθηκε η ανίχνευση δικτυακών ανωμαλιών. Μια άλλη εργασία [43] συνδύασε εξόρυξη δεδομένων ασαφούς λογικής και Γενετικούς αλγόριθμους τόσο για εύρεση ανωμαλιών (anomaly detection) όσο και για ανίχνευση προσβολών (misuse detection). Σε αυτή τη μελέτη, οι Γενετικοί Αλγόριθμοι χρησιμοποιήθηκαν για τον υπολογισμό παραμέτρων των ασαφών συναρτήσεων που χρησιμοποιήθηκαν καθώς και την επιλογή των πιο σχετικών δικτυακών χαρακτηριστικών για την ανίχνευση. Άλλη μια εργασία που χρησιμοποίησε Γενετικό Προγραμματισμό για την άντληση ταξινομικών κανόνων ήταν αυτή των Lu και Traore [44] η οποία χρησιμοποίησε το πλαίσιο υποστήριξης – εμπιστοσύνης (support – confidence framework) ως αντικειμενική συνάρτηση και χρησιμοποίησε ιστορικά δεδομένα για την εξόρυξη των κανόνων. Γενετικούς Αλγόριθμους υβριδικά με την τεχνική KNN (k-nearest neighbor) χρησιμοποίησαν και οι Melanie Middlemiss και Grant Dick [45] στην επιλογή των πιο σημαντικών χαρακτηριστικών για την αναγνώριση διαφορετικών ειδών επιθέσεων. Τέλος, άλλη μια υβριδική μέθοδος παρουσιάστηκε από τους Xia, T. κ.α. [46] το 2005 όπου η θεωρία πληροφορίας συνδυάστηκε με Γενετικό Αλγόριθμο δίνοντας πολύ καλά αποτελέσματα στο σύνολο δεδομένων KDD99 [1] ενώ μείωσε ταυτόχρονα το πλήθος των δικτυακών χαρακτηριστικών που χρησιμοποιήθηκαν για την εξαγωγή κανόνων.

Εκτός από επικουρικά, οι Γενετικοί Αλγόριθμοι έχουν χρησιμοποιηθεί απευθείας για την εξαγωγή κανόνων διαχωρισμού επιθέσεων. Ο Chittur στην εργασία του [47] χρησιμοποίησε Γενετικό Αλγόριθμο για ανίχνευση ανωμαλιών. Ο αλγόριθμος αυτός δημιουργούσε τυχαίους αριθμούς για τη δημιουργία συντελεστών βασισμένων στις εφήμερες τυχαίες σταθερές (Ephemeral Random Constants – ERC) οι οποίες έχουν περιγραφεί στο μαθηματικό μοντέλο του Koza [48]. Ο αλγόριθμος είχε πολύ καλά αποτελέσματα με μικρό ποσοστό κανονικών συνδέσεων κατηγοριοποιημένων ως εισβολών (false positives) αλλά χρησιμοποιούσε όλη την πληροφορία των δικτυακών χαρακτηριστικών με αποτέλεσμα την αργή σύγκλιση.

Μια άλλη προσέγγιση είναι αυτή του Li [49] στην οποία χρησιμοποιήθηκαν εννιά δικτυακά χαρακτηριστικά στην εξαγωγή κανόνων ανίχνευσης εισβολών τα οποία έλαβαν μέρος με αντίστοιχες βαρύτητες το καθένα. Αντίστοιχα έχουν προταθεί και άλλες προσεγγίσεις χρησιμοποιώντας σχετικά μικρό αριθμό δικτυακών χαρακτηριστικών [50], [51]. Ένα πλεονέκτημα της επιλογής αυτής είναι η γρήγορη ενημέρωση με νέους κανόνες ώστε, στον πραγματικό κόσμο να μπορούν να ανταποκριθούν στις συνεχόμενες νέου είδους επιθέσεις στο δίκτυο. Μια άλλη εφαρμογή [52], σχεδιάστηκε να παρακολουθεί τις δραστηριότητες στο δίκτυο σε διάφορα επίπεδα (χρήστη, συστήματος, διαδικασίας και πακέτου) και με τη βοήθεια Γενετικού ταξινομητή αποφασίζει για το αν υπάρχει παραβίαση ασφάλειας. Τέλος, οι Banković κ.α. [2], [53] με πολύ μικρό μέγεθος δικτυακών χαρακτηριστικών, εξάγουν κανόνες με τους Γενετικούς Αλγόριθμους που παρουσιάζει. Τα εξαγόμενα χαρακτηριστικά τα εξάγει με την τεχνική «Principal Component Analysis – PCA» και εφαρμόζει έναν κλασσικό Γενετικό Αλγόριθμο για την εξαγωγή κανόνων εισβολής.

Οι περισσότερες εφαρμογές, από το 2000 και έπειτα χρησιμοποιούν για την αξιολόγησή τους το σύνολο δικτυακών δεδομένων του διαγωνισμού KDD99 [1]. Στον παρακάτω πίνακα (Πίνακας 6), έχουν

καταγραφεί [2] τεχνικές μηχανικής μάθησης (machine learning) που έχουν αναπτυχθεί για την ανίχνευση εισβολών και τα αποτελέσματα των μετρήσεών τους στο KDD99 σύνολο δεδομένων.

Technique	Detection rate (%)	False positive rate (%)
C4.5	95	1
Support vector machine (SVM)	95.5	1
Multi layer perception (MLP)	94.5	1
k-nearest neighbor (k-NN)	92	1
Linear programming machine (LPM)	94	1
Regularized discriminant analysis (RDA)	94	1
Ficher linear discriminant (FD)	89	1
γ-algorithm	80	1
k-means clustering	65	1
Single leakage clustering	69	1
Quarter-sphere SVM	65	1
Y-means clustering	89.89	1
Genetic programming ensemble for distributed intrusion detection (GEdIDS)	91	0.43
SVM + GA	99	–
SVM + Fuzzy Logic	99.56	0.44
Neural Networks + PCA	92.22	–
C4.5 + PCA	92.16	–
GA	97.47	0.69
C4.5+Hybrid neural networks	93.28	0.2
Hidden Markov model (HMM)	79	–

Πίνακας 6: Τεχνικές γνώσης μηχανής και απόδοση τους στην εξόρυξη εισβολών

Είναι φανερό μέσα από την παραπάνω ανάλυση ότι το πεδίο των Γενετικών Αλγορίθμων στην ασφάλεια υπολογιστών είναι ανερχόμενο, τουλάχιστον όσον αφορά το μοντέλο των συστημάτων ανίχνευσης εισβολών που εξετάζεται μέσα από αυτή την εργασία.

4.2. Προσέγγιση ενός εξελικτικού αλγόριθμου στην εξόρυξη δικτυακών εισβολών

Η προσέγγιση του Εξελικτικού μοντέλου στην παρούσα υλοποίηση έχει ως σκοπό την υλοποίηση ενός Γενετικού Αλγόριθμου ο οποίος μέσα σε ένα σύνολο ιστορικών δικτυακών δεδομένων θα εξάγει κανόνες που αντιπροσωπεύουν εισβολές ώστε να χρησιμοποιηθούν έπειτα για το φιλτράρισμα εισβολών. Για να επιτευχθεί αυτό, περιλαμβάνει δυο στάδια. Το πρώτο στάδιο στο οποίο επικεντρώνεται η περισσότερη εργασία, είναι η εξόρυξη κανόνων εισβολής μέσα από ένα σύνολο δεδομένων που περιέχει αναγνωρισμένες εισβολές. Αποτελεί την εκπαίδευση του αλγορίθμου στο να αναγνωρίζει ανεπιθύμητες συνδέσεις. Έπειτα οι κανόνες που εξάγονται, χρησιμοποιούνται στην ανίχνευση εισβολών μέσα σε ένα μεγαλύτερο σύνολο δικτυακών δεδομένων ώστε να αξιολογηθεί η επίδοση του αλγορίθμου.

Στις επόμενες υποενότητες περιγράφονται οι συνιστώσες και η σχεδίαση του αλγορίθμου ώστε να εκπαιδευτεί και να εξάγει κανόνες εισβολής. Μόλις εξαχθούν οι κανόνες η ανίχνευση εισβολών και οι μετρήσεις είναι απλή και γρήγορη διαδικασία.

4.3. Ορισμός του προβλήματος

Το πρόβλημα που επιχειρείται να λυθεί είναι η ανίχνευση εισβολών μέσα σε ένα δίκτυο. Για να επιτευχθεί η ανίχνευση, χρειάζονται κάποιοι κανόνες στα χαρακτηριστικά του δικτύου ώστε ανάλογα με τις τιμές των χαρακτηριστικών να αναγνωρίζονται οι πιθανές εισβολές με τη μεγαλύτερη δυνατή ακρίβεια. Για παράδειγμα ένας κανόνας είναι:

Αν *(duration = "0" και protocol = "finger" και source_port = 8080 και source_ip = "195.251.160.4")*

Τότε *«Εισβολή»,*

όπου μέσα στη συνθήκη επιλογής υπάρχουν ορισμένα δικτυακά χαρακτηριστικά που όταν οι τιμές τους κάνουν την συνθήκη αληθή, τότε το σύστημα θεωρεί πως υπάρχει εισβολή.

Ο προτεινόμενος αλγόριθμος εξόρυξης εισβολών συνεπώς, πρέπει να εκπαιδεύεται ώστε να παράγει τέτοιους κανόνες. Για να εκπαιδευτεί, χρειάζεται ένα σύνολο δεδομένων όπου είναι γνωστές οι εισβολές και οι κανονικές συνδέσεις έτσι ώστε να μπορεί να αυτό-αξιολογηθεί και από γενιά σε γενιά να παράγει καλύτερους κανόνες.

Για την αξιολόγησή του πρέπει να ορισθούν ορισμένα μετρήσιμα μεγέθη. Στην περίπτωση της ανίχνευσης δικτυακών εισβολών μας ενδιαφέρει το ποσοστό ανίχνευσης (detection rate) καθώς και το ποσοστό ομαλών συνδέσεων οι οποίες χαρακτηρίστηκαν ως εισβολές (false positive rate).

4.4. Αναπαράσταση δεδομένων (Data representation)

Στην παρούσα σχεδίαση του Γενετικού Αλγορίθμου στην ανίχνευση εισβολών δόθηκε σημασία τα χαρακτηριστικά που θα επιλεχθούν για να συμμετέχουν ως κανόνες στην εξόρυξη εισβολών να είναι όσο το δυνατόν λιγότερα ώστε να επιτευχθεί ανίχνευση με τη λιγότερη δυνατή πληροφορία. Επίσης τα δεδομένα για την εκπαίδευση να μην χρειάζεται να είναι πολλά, έτσι ώστε ο αλγόριθμος να συγκλίνει όσο το δυνατόν πιο γρήγορα στην εξαγωγή κανόνων με τον κίνδυνο όμως να μην έχει το απαιτούμενο πλήθος πληροφορίας ώστε να ελέγξει σε μεγάλο βάθος το χώρο αναζήτησης. Στην επόμενη υπο-ενότητα περιγράφεται το σύνολο δεδομένων που χρησιμοποιήθηκε για να επιλεχθούν τα δικτυακά χαρακτηριστικά, να εκπαιδευτεί και να αξιολογηθεί ο αλγόριθμος.

4.4.1. Το σύνολο δεδομένων KDD99 (KDD99 dataset)

Το σύνολο δεδομένων KDD99 προέρχεται από τον διαγωνισμό εύρεσης εισβολών (Classifier Learning Contest) του γκρουπ της ACM για την ανακάλυψη γνώσης και εξόρυξης δεδομένων (ACM Special Interest Group on Knowledge Discovery and Data Mining) και στα πλαίσια του 5^{ου} συνεδρίου του γκρουπ (5th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining 1999) [54], [55]. Ο σκοπός ήταν να φτιαχτεί ένα σύστημα ταξινομητής (learning classifier) το οποίο να διαχωρίζει «κακές» συνδέσεις, δηλάδή εισβολές ή επιθέσεις και «κανονικές» συνδέσεις.

Το σύνολο δεδομένων παρασκευάστηκε στα εργαστήρια Lincoln του MIT σε συνεργασία με την υπηρεσία DARPA (Defense Advanced Research Projects Agency) από πραγματικά δεδομένα που συλλέχθηκαν σε πραγματικό περιβάλλον. Συλλέχθηκαν ακατέργαστα TCP dump δεδομένα (raw data) εννιά εβδομάδων σε ένα τυπικό περιβάλλον τοπικού δικτύου της πολεμικής αεροπορίας των Ηνωμένων Πολιτειών Αμερικής και το εμπλούτισαν με πολλαπλές επιθέσεις. Τα δεδομένα εκπαίδευσης που συγκεντρώθηκαν ήταν περίπου 5 εκατομμύρια εγγραφές και τα δεδομένα αξιολόγησης ήταν περίπου 2 εκατομμύρια εγγραφές. Στον παρακάτω πίνακα (Πίνακας 7), καταγράφονται το σύνολο των χαρακτηριστικών κάθε εγγραφής στα δεδομένα, τα χαρακτηριστικά δηλαδή κάθε δικτυακής σύνδεσης.

Όνομα χαρακτηριστικού	Περιγραφή	Τύπος δεδομένου
duration	length (number of seconds) of the connection	continuous
protocol_type	type of the protocol, e.g. tcp, udp, etc.	discrete
service	network service on the destination, e.g., http, telnet, etc.	discrete
src_bytes	number of data bytes from source to destination	continuous
dst_bytes	number of data bytes from destination to source	continuous
flag	normal or error status of the connection	discrete
land	1 if connection is from/to the same host/port; 0 otherwise	discrete
wrong_fragment	number of ``wrong'' fragments	continuous

urgent	number of urgent packets	continuous
hot	number of ``hot'' indicators	continuous
num_failed_logins	number of failed login attempts	continuous
logged_in	1 if successfully logged in; 0 otherwise	discrete
num_compromised	number of ``compromised'' conditions	continuous
root_shell	1 if root shell is obtained; 0 otherwise	discrete
su_attempted	1 if ``su root'' command attempted; 0 otherwise	discrete
num_root	number of ``root'' accesses	continuous
num_file_creations	number of file creation operations	continuous
num_shells	number of shell prompts	continuous
num_access_files	number of operations on access control files	continuous
num_outbound_cmds	number of outbound commands in an ftp session	continuous
is_host_login	1 if the login belongs to the ``hot'' list; 0 otherwise	discrete
is_guest_login	1 if the login is a ``guest''login; 0 otherwise	discrete
count	number of connections to the same host as the current connection in the past two seconds	continuous
serror_rate	% of connections that have ``SYN'' errors	continuous
rerror_rate	% of connections that have ``REJ'' errors	continuous
same_srv_rate	% of connections to the same service	continuous
diff_srv_rate	% of connections to different services	continuous
srv_count	number of connections to the same service as the current connection in the past two seconds	continuous
srv_serror_rate	% of connections that have ``SYN'' errors	continuous
srv_rerror_rate	% of connections that have ``REJ'' errors	continuous
srv_diff_host_rate	% of connections to different hosts	continuous
dst_host_count	count of connections having the same destination host	continuous
dst_host_srv_count	count of connections having the same destination host and using the same service	continuous
dst_host_same_srv_rate	% of connections having the same destination host and using the same service	continuous
dst_host_diff_srv_rate	% of different services on the current host	continuous
dst_host_same_src_port_rate	% of connections to the current host having the same src port	continuous
dst_host_srv_diff_host_rate	% of connections to the same service coming from different host	continuous
dst_host_serror_rate	% of connections to the current host that have S0 error	continuous
dst_host_srv_serror_rate	% of connections to the current host and specified service that have an S0 error	continuous
dst_host_rerror_rate	% of connections to the current host that have RST errors	continuous
dst_host_srv_rerror_rate	% of connections to the current host and specified service that have an RST error	continuous

Πίνακας 7: Λίστα χαρακτηριστικών και περιγραφή τους στο σύνολο δεδομένων KDD99

Εξ ορισμού στους Γενετικούς αλγόριθμους η αναπαράσταση ενός χρωμοσώματος μέσα σε ένα πληθυσμό μιας γενιάς είναι και μια πιθανή λύση του προβλήματος. Στην συγκεκριμένη προσέγγιση κάθε χρωμόσωμα είναι και ένας πιθανός κανόνας που αναγνωρίζει εισβολή. Κάθε γονίδιο (gene) μέσα σε ένα χρωμόσωμα αντιπροσωπεύει και ένα χαρακτηριστικό δικτυακής σύνδεσης. Πρώτο βήμα είναι η επιλογή του πλήθους των χαρακτηριστικών που θα συμμετέχουν ως γονίδια μέσα σε ένα χρωμόσωμα. Τα χαρακτηριστικά που επιλέχθηκαν στην παρούσα προσέγγιση ήταν αντίστοιχα με αυτά της Bankovic [2] όπου χρησιμοποιώντας τη μέθοδο PCA (Principal Component Analysis), μια τεχνική μείωσης διαστάσεων και πιο συγκεκριμένα το μετασχηματισμό Karhunen–Loève [56], αλλά και μια άλλη μέθοδο πολυμεταβλητής ανάλυσης [57], εξήγαγε ανά πλήθος χαρακτηριστικών, τα χαρακτηριστικά με την μεγαλύτερη διακύμανση (variance) (Πίνακας 8). Με τον τρόπο αυτό επιλέγοντας τα σημαντικότερα χαρακτηριστικά, μειώνεται το κόστος υπολογισμών των όχι τόσο σημαντικών χαρακτηριστικών και η ταχύτητα εκπαίδευσης και εξόρυξης εισβολών αυξάνεται.

Dimension	1	2	3	4	5	6	7	8
	src_bytes	duration src_bytes	duration src_bytes dst_host _srv_ serror_rate	duration src_bytes serror_rate	duration src_bytes serror_rate	duration flag src_bytes	duration flag src_bytes	duration flag src_bytes
				dst_host _srv_ serror_rate	dst_bytes	serror_rate	hot	srv_rerror_rate
					dst_host _srv_ serror_rate	dst_bytes	serror_rate	hot
						dst_host _srv _serror_rate	dst_bytes	serror_rate
							dst_host _srv_ serror_rate	dst_bytes
								dst_host_srv_ serror_rate

Πίνακας 8: Λίστα με τα σημαντικότερα χαρακτηριστικά ανά διάσταση [2]

4.4.2. Αναπαράσταση χρωμοσωμάτων

Κρατώντας όσο το δυνατόν πιο μικρό τον αριθμό των χαρακτηριστικών, προσπαθώντας παράλληλα να διατηρηθεί υψηλό ποσοστό ανίχνευσης, η διάσταση που επιλέχθηκε ήταν τριών χαρακτηριστικών (Πίνακας 9) οπότε και το κάθε χρωμόσωμα θα αποτελείται από 3 γονίδια.

Χαρακτηριστικό	**Περιγραφή**	**Αριθμός γονιδίων**
duration	Διάρκεια σύνδεσης σε δευτερόλεπτα	1
src_bytes	Αριθμός bytes από την πηγή προς τον προορισμό	1
dst_host_srv_serror_rate	Ποσοστό των συνδέσεων που έχουν “SYN” σφάλμα	1

Πίνακας 9: Επιλεγμένα δικτυακά χαρακτηριστικά ως γονίδια

Η αναπαράσταση των χρωμοσωμάτων στον αλγόριθμο που σχεδιάστηκε είναι φαινοτυπική. Στην φαινοτυπική αναπαράσταση [58] τα χρωμοσώματα στον πληθυσμό αντιστοιχούν απευθείας στο χώρο δυνατών λύσεων σε αντίθεση με τη γονοτυπική η οποία χρησιμοποιείται στην κλασσική προσέγγιση ενός Γενετικού αλγορίθμου όπου η αναπαράσταση των γονιδίων και κατ’ επέκταση του

χρωμοσώματος, είναι δυαδική. Αυτό επηρεάζει και την επιλογή των γενετικών ανασυνδυασμών όπως αναλύεται στην επόμενη ενότητα για τους γενετικούς τελεστές του προτεινόμενου αλγόριθμου.

Έτσι, ένα χρωμόσωμα στον Αλγόριθμο αναπαρίσταται ως εξής: {2, 132, 3} όπου σημαίνει 2 δευτερόλεπτα για το πεδίο duration, 132 bytes για το πεδίο src_bytes και 3% ποσοστό για το πεδίο dst_host_srv_serror_rate. Συνεπώς ο κανόνας εισβολής για το παραπάνω χρωμόσωμα είναι:

Αν *(duration = 2 και* src_bytes *= 132 και* dst_host_srv_serror_rate *= 3)*

Τότε *«Εισβολή»*

Αποτέλεσμα της φαινοτυπικής αναπαράστασης είναι, κάθε γονίδιο στο χρωμόσωμα να αναπαρίσταται από ακέραιους αριθμούς. Το πεδίο ορισμού κάθε γονιδίου προέρχεται μέσα από τις δυνατές τιμές που μπορεί να πάρει μέσα από το σύνολο δεδομένων εκπαίδευσης του αλγορίθμου.

4.5. Η αντικειμενική συνάρτηση

Για την αντικειμενική συνάρτηση ή συνάρτηση αξιολόγησης (evaluation function) του αλγορίθμου ο βασικός σκοπός ήταν να αυξάνεται ανάλογα με το πόσες επιθέσεις σωστά αναγνωρίζει το εκάστοτε χρωμόσωμα στον πληθυσμό. Παράλληλα, η αντικειμενική συνάρτηση θα πρέπει να λαμβάνει υπόψη της και πόσες επιθέσεις αναγνώρισε λάθος ώστε τελικά να μη φτάσει να δίνει μεγάλη αξιολόγηση σε κανόνα εισβολής ο οποίος αναγνωρίζει αρκετές εισβολές παράλληλα όμως να ξεχωρίζει ως εισβολές και κανονικές (normal) συνδέσεις. Έτσι η συνάρτηση αξιολόγησης που επιλέχθηκε και η οποία έχει χρησιμοποιηθεί συχνά σε αντίστοιχες μελέτες [47], [2], [50] είναι η παρακάτω:

$$fitness = \frac{\alpha}{A} - \frac{\beta}{B}$$

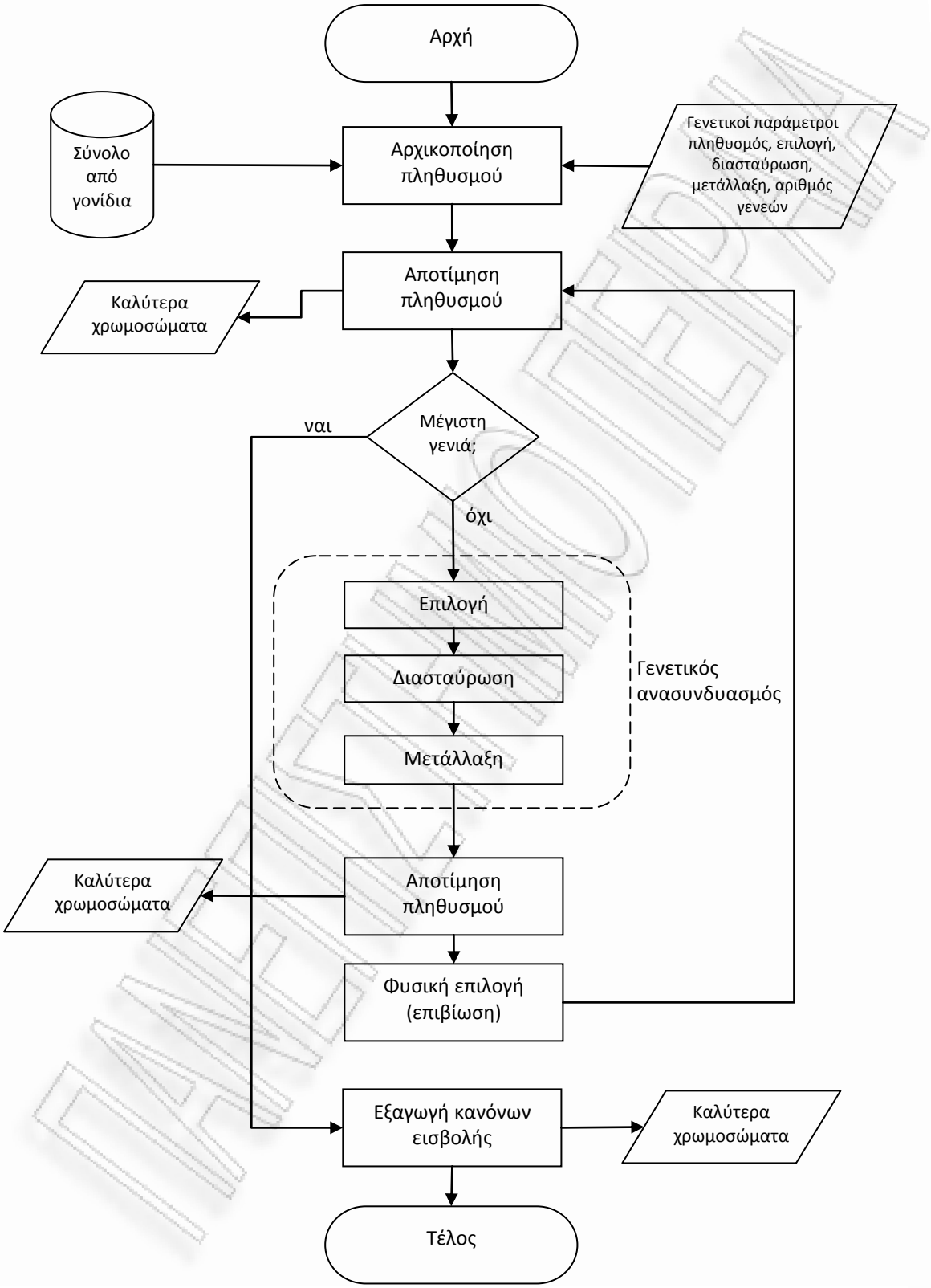
όπου α είναι ο αριθμός των συνδέσεων που αναγνωρίστηκαν σωστά ως εισβολές, Α είναι ο συνολικός αριθμός επιθέσεων στο σύνολο δεδομένων εκπαίδευσης, β είναι ο αριθμός των κανονικών συνδέσεων που λανθασμένα αναγνωρίστηκαν ως επιθέσεις (false positives) και Β είναι ο αριθμός των κανονικών συνδέσεων στο σύνολο δεδομένων εκπαίδευσης.

Το αποτέλεσμα της παραπάνω συνάρτησης ανήκει στο πεδίο [-1, 1] όπου -1 είναι η μικρότερη δυνατή τιμή (ο κανόνας αναγνώρισε όλες τις κανονικές συνδέσεις ως επιθέσεις και καμία σωστά αναγνωρισμένη επίθεση) και 1 η μέγιστη τιμή (ο κανόνας αναγνώρισε όλες τις επιθέσεις σωστά χωρίς να αναγνωρίσει καμία κανονική σύνδεση ως λανθασμένη). Υψηλό ποσοστό εντοπισμού επιθέσεων και παράλληλα μείωση των λανθασμένων αναγνωρίσεων σημαίνει μεγαλύτερη τιμή στην αντικειμενική συνάρτηση και το αντίστροφο.

4.6. Επισκόπηση του Αλγορίθμου

Στο Σχήμα 11, αναπαρίσταται ο αλγόριθμος με διάγραμμα ροής.

Στην αρχή ο αλγόριθμος αφού θέσει τις γενετικές παραμέτρους, το πλήθος των γενεών που θα τρέξει, τον αριθμό των χρωμοσωμάτων μέσα στον πληθυσμό, τον τρόπο με τον οποίο θα επιλέξει του καλύτερους γονείς για αναπαραγωγή, τις παραμέτρους διασταύρωσης και μετάλλαξης, θα δημιουργήσει τον αρχικό πληθυσμό με τυχαίο τρόπο, μέσα όμως από τις δυνατές τιμές που μπορεί να πάρει από το σύνολο δεδομένων εκπαίδευσης (training dataset). Αφού αποτιμηθεί ο αρχικός πληθυσμός μέσω της αντικειμενικής συνάρτησης, ο πληθυσμός εξελίσσεται από γενεά σε γενεά μέχρι το όριο των γενεών που έχει ορισθεί αρχικά.



Σχήμα 11: Το διάγραμμα ροής του προτεινόμενου Γενετικού Αλγορίθμου για την εξόρυξη εισβολών

Σε κάθε γενιά, γίνεται η επιλογή των καλύτερων κανόνων ανάλογα με την παράμετρο της επιλογής που έχει τεθεί (parent selection) και ακολουθεί η διασταύρωση και η μετάλλαξή τους. Στην προτεινόμενη προσέγγιση, αφού γίνει η αποτίμηση πληθυσμού ακολουθεί η επιλογή επιβίωσης του πληθυσμού (survival selection), καθώς έχει εισαχθεί στον αλγόριθμο η παράμετρος του μεγέθους επώασης (brood size). Αυτό σημαίνει πως ο πατρικός πληθυσμός ανάλογα με την παράμετρο μπορεί κατά την επιλογή του να δημιουργήσει διπλάσιους απογόνους (για brood size = 2) και έπειτα από αυτούς τους απογόνους μετά την επιβίωσή τους να προκύψει ο πληθυσμός της επόμενης γενιάς.

Εκτός των παραπάνω, η προτεινόμενη προσέγγιση σε κάθε αποτίμηση πληθυσμού καταγράφει ορισμένα διακριτά χρωμοσώματα τα οποία έχουν τιμή αντικειμενικής συνάρτησης > 0. Με αυτό τον τρόπο κρατάει ένα ιστορικό των «καλύτερων» κανόνων κατά τη διάρκεια της εξέλιξης των γενεών γιατί υπάρχει πιθανότητα σε κάποια γενιά να βρεθεί χρωμόσωμα με καλή απόδοση (>0) το οποίο στη έλευση των επόμενων γενεών, να χαθεί. Στο τέλος, όπου ο αλγόριθμος φτάνει το μέγιστο πλήθος γενεών, εξάγονται οι καλύτεροι κανόνες και ελέγχονται τόσο στο σύνολο δεδομένων της εκπαίδευσης όσο και στο σύνολο δεδομένων αξιολόγησης, για το ποσοστό εξόρυξης εισβολών.

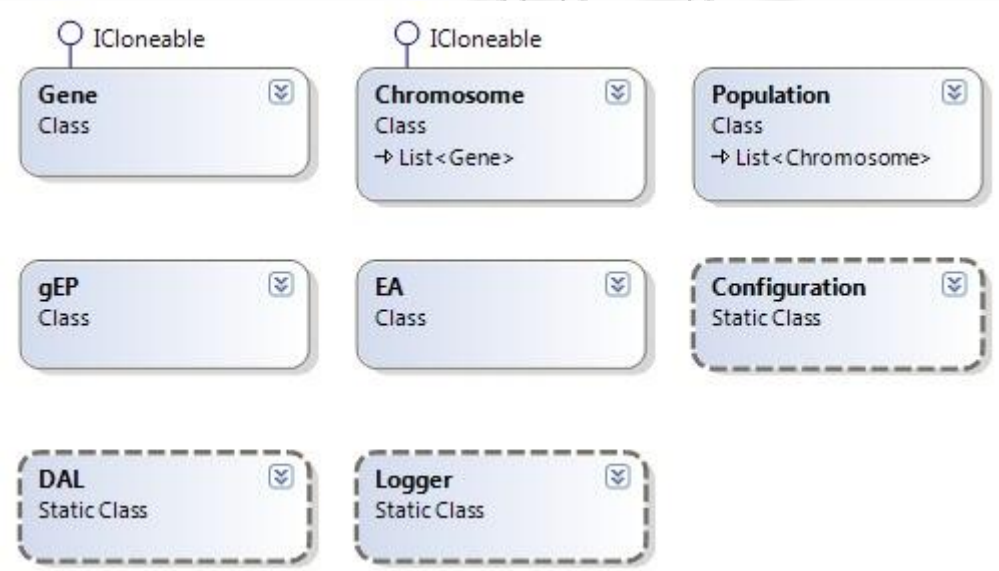
5. Υλοποίηση του συστήματος

Στην ενότητα αυτή περιγράφονται οι προδιαγραφές και τα τεχνικά χαρακτηριστικά με τα οποία υλοποιήθηκε ο Γενετικός Αλγόριθμος για την εκπαίδευση του συστήματος και την εξαγωγή κανόνων δικτυακών εισβολών.

Το πρόγραμμα γράφηκε με τη Γλώσσα Προγραμματισμού C# στο περιβάλλον του Microsoft Visual Studio (αρχικά στην έκδοση 2008 και έπειτα μεταφέρθηκε στο Visual Studio 2010). Για τη μεταγλώττιση αρκούσε το .NET Framework 2.0 οπότε και το τελικό εκτελέσιμο αρχείο που βρίσκεται στο συνοδευτικό κώδικα, απαιτεί το .NET Framework 2.0 για την εκτέλεσή του σε περιβάλλον MS Windows.

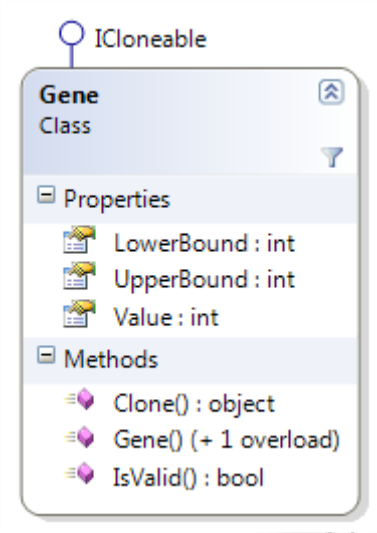
Επίσης, για την επικοινωνία με το περιβάλλον εκπαίδευσης, δηλαδή τις δικτυακές εισβολές που χρησιμοποιήθηκαν για την εξέλιξη του αλγορίθμου και την εξόρυξη των κανόνων, χρησιμοποιήθηκε βάση δεδομένων ώστε το σύνολο δεδομένων KDD του οποίου τα αρχεία ήταν σε comma delimited αρχεία κειμένου, να επεξεργαστεί ανάλογα με τα χαρακτηριστικά που θα χρησιμοποιούνταν από τον αλγόριθμο. Με τη χρήση βάσης δεδομένων ήταν εύκολο μετά την εξαγωγή κανόνων από τον αλγόριθμο, να αξιολογηθεί η ανίχνευση των εισβολών στο περιβάλλον αξιολόγησης. Το σύστημα διαχείρισης βάσης δεδομένων που χρησιμοποιήθηκε τόσο από το πρόγραμμα, για την επικοινωνία με τις τιμές των χαρακτηριστικών του συνόλου δεδομένων, όσο και για την αξιολόγηση των κανόνων εισβολής, ήταν ο MS SQL Server 2008.

Οι κλάσεις από τις οποίες αποτελείται το πρόγραμμα, απεικονίζονται στο Σχήμα 12:



Σχήμα 12: Το διάγραμμα των κλάσεων του υλοποιημένου συστήματος

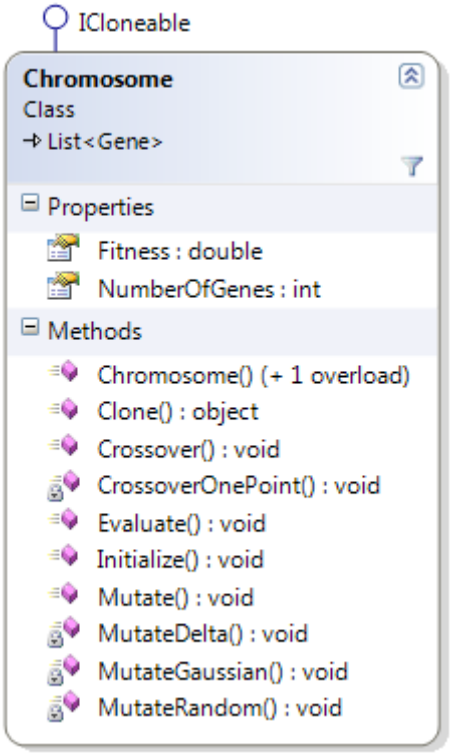
Η κλάση Gene, αντιπροσωπεύει ένα γονίδιο ενός χρωμοσώματος. Οι ιδιότητες και οι μέθοδοί της φαίνονται παρακάτω (Σχήμα 13).



Σχήμα 13: Οι ιδιότητες και οι μέθοδοι της κλάσης Gene

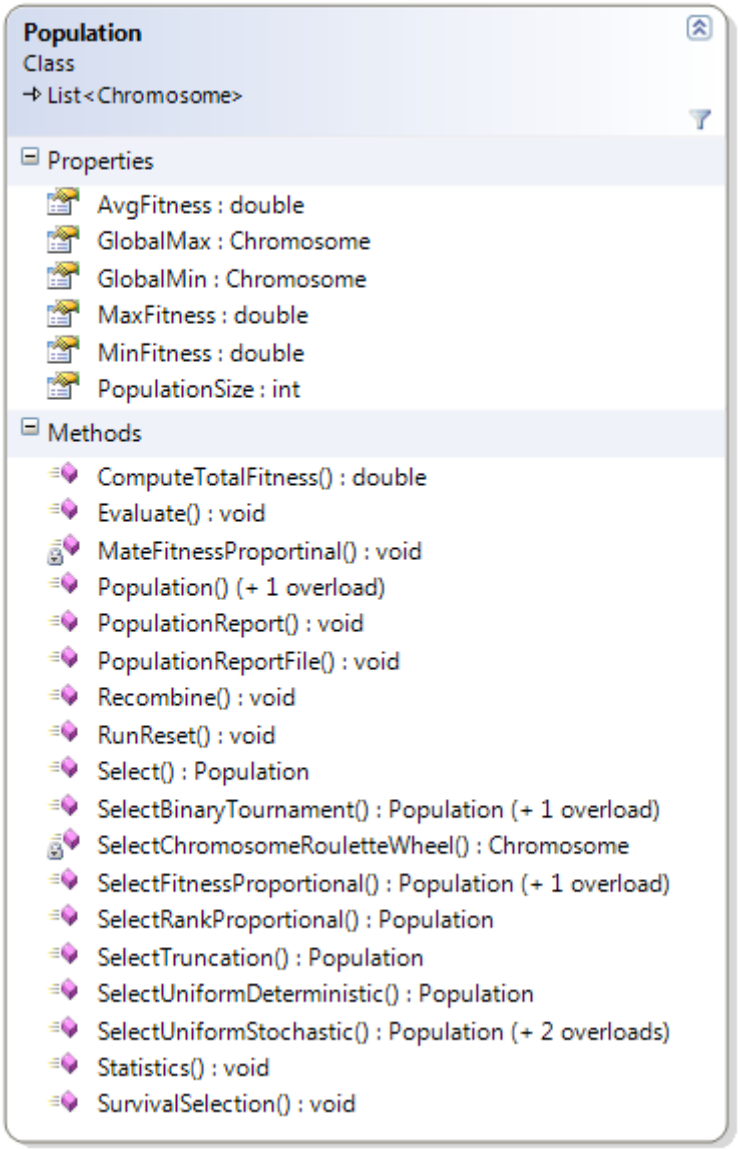
Ένα γονίδιο αποτελείται από την τιμή που έχει και από τη μέγιστη και ελάχιστη τιμή που μπορεί να πάρει. Η κλάση, εκτός από τη μέθοδο κατασκευής (constructor method) του γονιδίου, υλοποιεί και τη μέθοδο Clone() για να αντιγράφει τον εαυτό του σε ένα καινούριο γονίδιο.

Η κλάση Chromosome (Σχήμα 14), αντιπροσωπεύει ένα χρωμόσωμα ενός πληθυσμού και αποτελείται από μια λίστα από χρωμοσώματα (List<Gene>). Επιπλέον, έχει το χαρακτηριστικό Fitness το οποίο αντιπροσωπεύει την τιμή που έχει η συνάρτηση αξιολόγησης για το συγκεκριμένο χρωμόσωμα καθώς και τον αριθμό γονιδίων που περιέχει (NumberOfGenes). Στις μεθόδους που υλοποιεί, εκτός από τις μεθόδους δημιουργίας ενός χρωμοσώματος και την αντιγραφή του εαυτού του σε καινούριο, υλοποιεί και τις λειτουργίες της διασταύρωσης και μετάλλαξης στο εκάστοτε χρωμόσωμα. Η μέθοδος Evaluate() υπολογίζει και αποδίδει τιμή στο χαρακτηριστικό Fitness. Η μέθοδος Initialize() αρχικοποιεί τα γονίδια του χρωμοσώματος όταν κληθεί, ώστε να πάρουν ορισμένες αρχικές τιμές από τις δυνατές τιμές του υποσυνόλου δεδομένων KDD που χρησιμοποιείται για εκπαίδευση. Έτσι στην δημιουργία του πληθυσμού της πρώτης γενιάς, δεν δημιουργούνται εντελώς τυχαίες τιμές στα γονίδια του χρωμοσώματος, αλλά τιμές μέσα από τις δυνατές τιμές που μπορεί να πάρει κάθε χαρακτηριστικό του δικτύου (γονιδίου) αντίστοιχα.



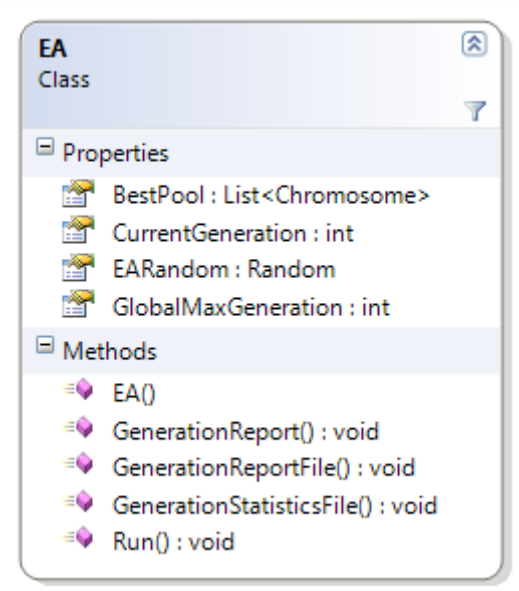
Σχήμα 14: Οι ιδιότητες και οι μέθοδοι της κλάσης Chromosome

Η κλάση Population (Σχήμα 15), αντιπροσωπεύει έναν πληθυσμό και αποτελείται από μια λίστα από χρωμοσώματα (List<Chromosome>), τα άτομα του πληθυσμού. Η κλάση έχει χαρακτηριστικά που αφορούν τόσο τον πληθυσμό, όσο και στοιχεία που αφορούν τα στατιστικά που χρειάζονται για τον έλεγχο του αλγορίθμου. Αυτά περιλαμβάνουν το μέγεθος του πληθυσμού (PopulationSize), την μέση απόδοση του πληθυσμού (AvgFitness), τη μέγιστη και ελάχιστη απόδοση που έχει ο εκάστοτε πληθυσμός, καθώς τα δύο χρωμοσώματα που έχουν τη μέγιστη και ελάχιστη απόδοση μέχρι τώρα από την αρχή της εξέλιξης των γενεών. Οι μέθοδοι που υλοποιεί η κλάση, αφορούν αφενός τον υπολογισμό της απόδοσης κάθε χρωμοσώματος του πληθυσμού αφετέρου τις μεθόδους για την επιλογή χρωμοσωμάτων από τον πληθυσμό. Η επιλογή περιλαμβάνει τόσο την επιλογή γονέων ανάλογα με τις παραμέτρους που έχουν δοθεί αρχικά στον αλγόριθμο, καθώς και την επιλογή για επιβίωση, τα χρωμοσώματα που θα περάσουν στην επόμενη γενιά για αναπαραγωγή. Τέλος περιλαμβάνει και τη μέθοδο Statistics(), η οποία χρησιμοποιείται για να εξάγονται στατιστικά από τον πληθυσμό και να καταγράφονται σε αρχείο.



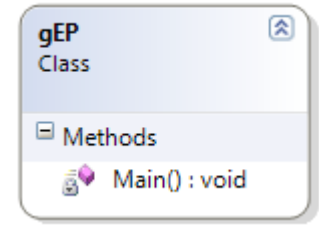
Σχήμα 15: Οι ιδιότητες και οι μέθοδοι της κλάσης Population

Η κλάση EA (Σχήμα 16), αντιπροσωπεύει ένα εξελικτικό αλγόριθμο (Evolution Algorithm) ο οποίος, ανάλογα με το πόσες φορές θέλουμε να τρέξει ο Αλγόριθμος, δημιουργείται και υλοποιεί την εξέλιξη των γενεών μέχρι την μέγιστη γενιά που έχει οριστεί. Οι ιδιότητες της κλάσης περιλαμβάνουν μια λίστα από τα καλύτερα χρωμοσώματα που μέχρι τώρα έχουν παραχθεί (BestPool) με σκοπό στο τέλος από αυτή τη λίστα να εξαχθούν και οι κανόνες εισβολής. Επίσης αποθηκεύει και σε ποια γενιά βρίσκεται ο αλγόριθμος (CurrentGeneration), σε ποια γενιά βρέθηκε το καλύτερο χρωμόσωμα (GlobalMaxGeneration), καθώς και ένα αντικείμενο τύπου Random (EARandom) για την παραγωγή ψευδό-τυχαίων αριθμών, όποτε χρειαστεί η τυχαιότητα στον αλγόριθμο. Η τυχαιότητα προέρχεται με δύο τρόπους στο πρόγραμμα, ανάλογα με τις ρυθμίσεις των παραμέτρων: είτε χρησιμοποιώντας την ώρα ως τυχαία τιμή παραγωγής (seed value), είτε χρησιμοποιώντας μια συγκεκριμένη τιμή παραγωγής. Η βασική μέθοδος της κλάσης EA είναι η Run(), η οποία προσομοιώνει όλη την εξέλιξη των γενεών μέσα από ένα αρχικό πληθυσμό. Επαναληπτικά, δημιουργεί τους απογόνους κάθε γενιάς, τους αναπαράγει, εκτελεί την επιλογή επιβίωσης, τους απογόνους που θα επιβιώσουν στην επόμενη γενιά και καλεί τον εκάστοτε πληθυσμό να αποτιμήσει την απόδοση των χρωμοσωμάτων του. Παράλληλα υπολογίζει και δημιουργεί αναφορές για τον πληθυσμό κάθε γενιάς σε αρχείο.



Σχήμα 16: Οι ιδιότητες και οι μέθοδοι της κλάσης EA

Η κλάση gEP (Σχήμα 17), αποτελεί το σημείο έναρξης εκτέλεσης του προγράμματος και του Γενετικού Αλγορίθμου με τη μέθοδο Main(). Εκτός από την έναρξη του Αλγορίθμου και πριν από αυτή, φορτώνει τις παραμέτρους που αφορούν τόσο το πρόγραμμα, όσο και τους γενετικούς τελεστές.



Σχήμα 17: Η κλάση gEP

Για την φόρτωση των παραμέτρων εκτέλεσης, το πρόγραμμα χρησιμοποιεί ένα XML αρχείο με το όνομα “gEP.xml” το οποίο περιμένει να είναι στον ίδιο φάκελο, μαζί με το εκτελέσιμο αρχείο του προγράμματος. Σε αυτό το αρχείο δηλώνονται οι παράμετροι με τις οποίες κάθε φορά εκτελείται ο αλγόριθμος. Παρακάτω επισυνάπτεται και επεξηγείται το περιεχόμενο ενός αρχείου “gEP.xml”:

```
<?xml version="1.0" encoding="utf-8"?>
<EP>
	<Runs>1</Runs>
	<RandomSeed enabled="True">1351</RandomSeed>
	<PopulationSize>100</PopulationSize>
	<BroodSize>2</BroodSize>
	<Generations>300</Generations>
	<Genes>
		<Gene bounds="True">
			<min>0</min>
			<max>42088</max>
		</Gene>
		<Gene bounds="True">
			<min>0</min>
			<max>15377</max>
		</Gene>
		<Gene bounds="True">
```

```
                    <min>0</min>
                    <max>100</max>
                </Gene>
        </Genes>
        <Crossover type="ONE_POINT">0,7</Crossover>
        <Mutation type="RANDOM_MUTATION" rate="one">1</Mutation>
        <ParentSelection>BINARY_TOURNAMENT</ParentSelection>
        <SurvivalSelection>FITNESS_PROPORTIONAL</SurvivalSelection>
        <ConnectionString>Data Source=localhost;Initial Catalog=KDD99;Integrated
Security=True</ConnectionString>
</EP>
```

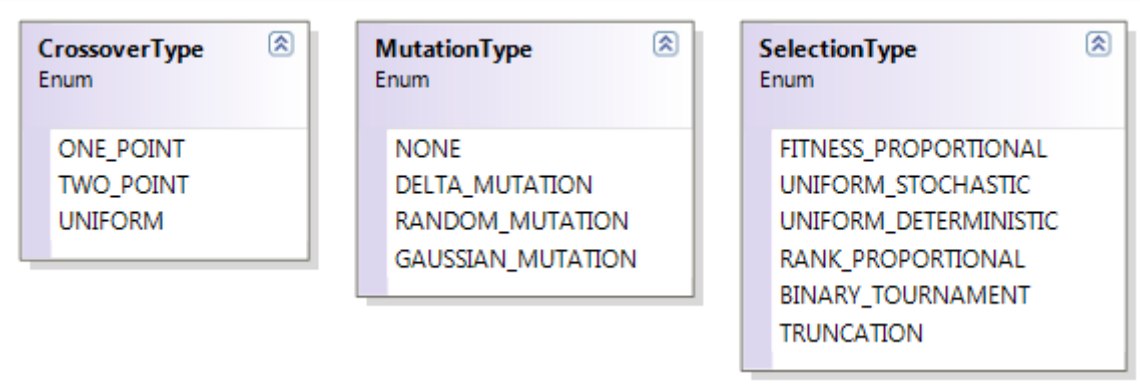
Οι παράμετροι εσωκλείονται σε μια κεντρική ετικέτα (xml tag) η οποία ονομάζεται <EP>. Η ετικέτα <Runs> περιέχει τον αριθμό των φορών που θέλουμε να τρέξουμε τον Αλγόριθμο. Η ετικέτα <RandomSeed> εάν έχει το χαρακτηριστικό (attribute) enabled="True" θα χρησιμοποιήσει τον ακέραιο αριθμό του περιεχόμενού της ως βάση (seed) για παραγωγή ψευδό-τυχαίων αριθμών. Στην περίπτωση που είναι enabled="False" θα χρησιμοποιήσει την ώρα για παραγωγή ψευδό-τυχαίων αριθμών.

Η ετικέτα <PopulationSize> περιέχει το πλήθος των ατόμων – χρωμοσωμάτων που θα έχει ο πληθυσμός στον Γενετικό Αλγόριθμο καθ' όλη τη διάρκεια παρέλευσης των γενεών. Η ετικέτα <BroodSize> περιέχει το μέγεθος αναπαραγωγής των γονέων, δηλαδή πόσους απογόνους αναπαράγει κάθε γονέας που επιλέγεται για αναπαραγωγή. Η ετικέτα <Generations> περιέχει τον αριθμό των γενεών για τις οποίες θα εκτελεστεί ο αλγόριθμος, δηλαδή πόσες φορές ο αρχικός πληθυσμός θα αναπαραχθεί.

Η ετικέτα <Genes> περιέχει τόσες ετικέτες <Gene> όσες είναι και τα γονίδια τα οποία υπάρχουν σε κάθε χρωμόσωμα. Αν η ετικέτα <Gene> έχει το χαρακτηριστικό bounds="True", τότε στις ετικέτες <min> και <max> καθορίζονται τα ελάχιστα και τα μέγιστα όρια τιμών που μπορεί να λάβει το κάθε γονίδιο στο χρωμόσωμα, αντίστοιχα.

Η ετικέτα <Crossover> περιέχει τον τύπο διασταύρωσης που θα χρησιμοποιήσει ο αλγόριθμος στο χαρακτηριστικό type="" και την πιθανότητα με την οποία θα συμβαίνει, στο περιεχόμενο της ετικέτας. Οι τύποι διασταύρωσης που μπορούν να χρησιμοποιηθούν στο αρχείο ρυθμίσεων παραμέτρων είναι "ONE_POINT", "TWO_POINT" και "UNIFORM". Ο αλγόριθμος υλοποιεί διασταύρωση ενός σημείου τα υπόλοιπα υποστηρίζονται για μελλοντική επέκταση του αλγορίθμου.

Η ετικέτα <Mutation> περιέχει το είδος και την πιθανότητα μετάλλαξης στον αλγόριθμο. Τα είδη μετάλλαξης φαίνονται στο Σχήμα 18. Το χαρακτηριστικό γνώρισμα rate μπορεί να πάρει τις εξής τρείς τιμές: "one", "all", "probability". Όταν η τιμή του είναι "one" αυτό σημαίνει στατιστικά θα μεταλλάσσεται ένα γονίδιο από ολόκληρο το χρωμόσωμα, αν για παράδειγμα έχουμε χρωμόσωμα με 5 γονίδια τότε ουσιαστικά η πιθανότητα μετάλλαξης θα είναι 1/5 = 0,2. Όταν η τιμή είναι "all", τότε όλα τα γονίδια θα μεταλλάσσονται και η πιθανότητα μετάλλαξης ουσιαστικά θα είναι 1,0. Τέλος, αν η τιμή είναι "probability", τότε η πιθανότητα μετάλλαξης θα είναι το περιεχόμενο της ετικέτας <Mutation>.



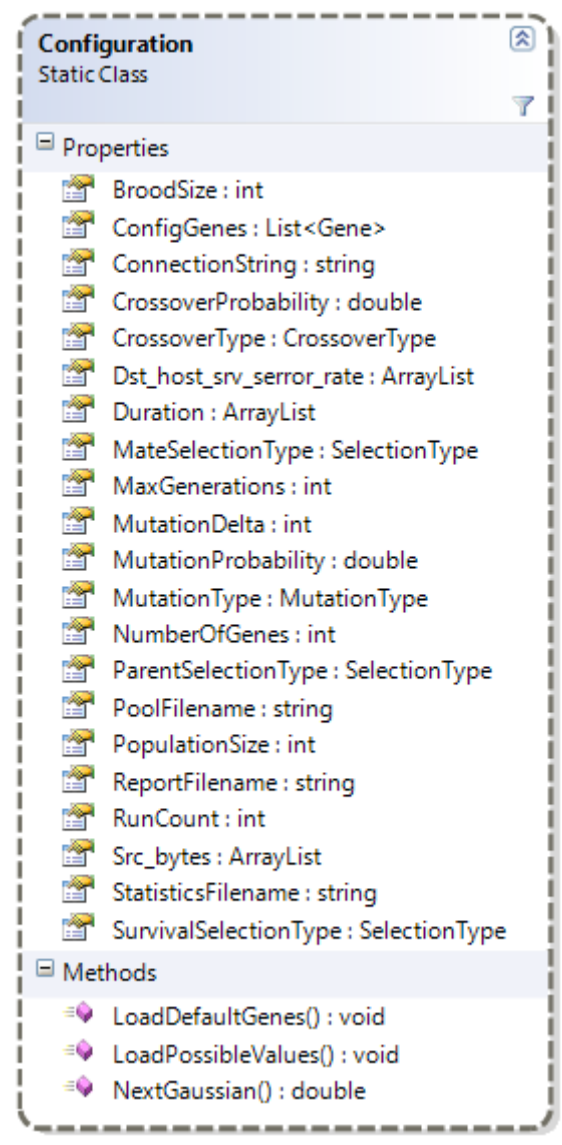
Σχήμα 18: Οι απαριθμητές (enumerators) που αφορούν την διασταύρωση, μετάλλαξη και επιλογή αντίστοιχα.

Η ετικέτα `<ParentSelection>` περιέχει το είδος επιλογής που θα χρησιμοποιηθεί κατά την επιλογή γονέων προς αναπαραγωγή. Η υλοποίηση του αλγορίθμου υποστηρίζει την δυαδική επιλογή τουρνουά (Binary Tournament), την επιλογή ρουλέτας (Fitness proportional) και την ομοιόμορφη στοχαστική (Uniform stochastic) όσον αφορά την επιλογή γονέων προς αναπαραγωγή.

Η ετικέτα `<SurvivalSelection>` περιέχει το είδος επιλογής που θα χρησιμοποιηθεί για την επιβίωση των απογόνων των γονέων. Η υλοποίηση του αλγορίθμου υποστηρίζει την ομοιόμορφη στοχαστική και την επιλογή ρουλέτας όσον αφορά την επιλογή για την επιβίωση των απογόνων.

Τέλος, η ετικέτα `<ConnectionString>` περιέχει τα στοιχεία για τη σύνδεση με τη βάση δεδομένων, όνομα βάσης, δικτυακή τοποθεσία, στοιχεία λογαριασμού σύνδεσης.

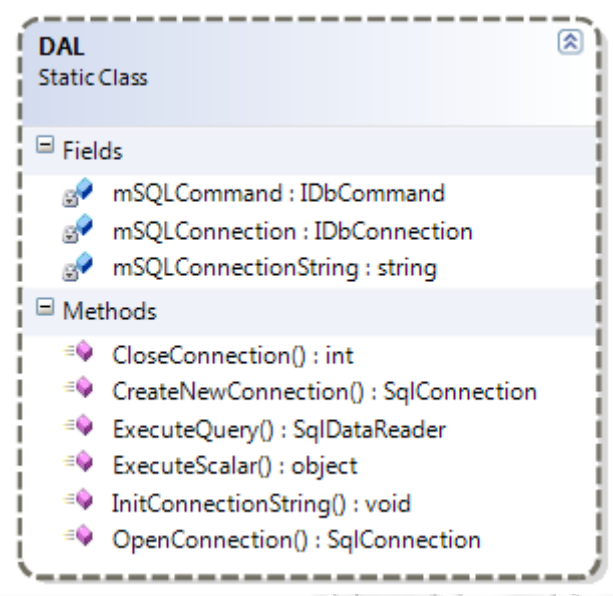
Η κλάση Configuration (Σχήμα 19), είναι μια στατική κλάση, έχει δηλαδή μόνο στατικά χαρακτηριστικά [59], η οποία περιέχει τις ρυθμίσεις που αφορούν όλο τον αλγόριθμο και οι ιδιότητές της αρχικοποιούνται κατά την ανάγνωση του αρχείου “gEP.xml” που περιέχει τις παραμέτρους. Έτσι, κάνει διαθέσιμες τις ρυθμίσεις του αλγορίθμου καθόλη τη διάρκεια εκτέλεσης του προγράμματος.



Σχήμα 19: Οι ιδιότητες και μέθοδοι της κλάσης Configuration

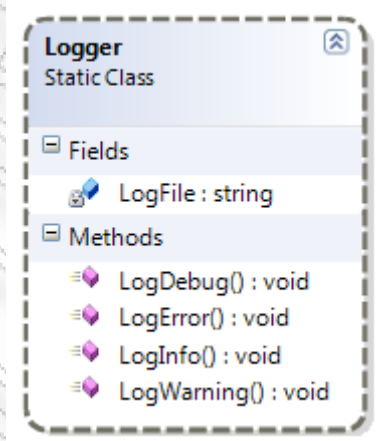
Η κλάση Configuration περιέχει και μια μέθοδο η οποία εκτελείται πριν ξεκινήσει το τμήμα του αλγορίθμου της εξέλιξης των γενεών, την LoadPossibleValues(). Η λειτουργία της μεθόδου αυτή είναι να φορτώσει στην μνήμη για κάθε ένα γονίδιο, τις πιθανές τιμές που μπορεί να πάρει, ουσιαστικά το πεδίο τιμών του κάθε γονιδίου. Έτσι, στην συνέχεια, όταν χρειαστεί να δοθούν τιμές κατά τη διάρκεια της αναπαραγωγής και της εφαρμογής γενικότερα των γενετικών τελεστών, οι τιμές θα επιλεχθούν από αυτό το σύνολο τιμών και όχι με νέο ερώτημα στη βάση. Η διαχείριση ορισμένων δεδομένων στη μνήμη (in-memory caching) βελτιώνει και την ταχύτητα του προγράμματος.

Η κλάση DAL (Σχήμα 20), είναι και αυτή μια στατική κλάση η οποία περιέχει τις μεθόδους για την σύνδεση και πρόσβαση δεδομένων της βάσης δεδομένων. Αποτελεί το επίπεδο πρόσβασης δεδομένων (Data Access Layer).



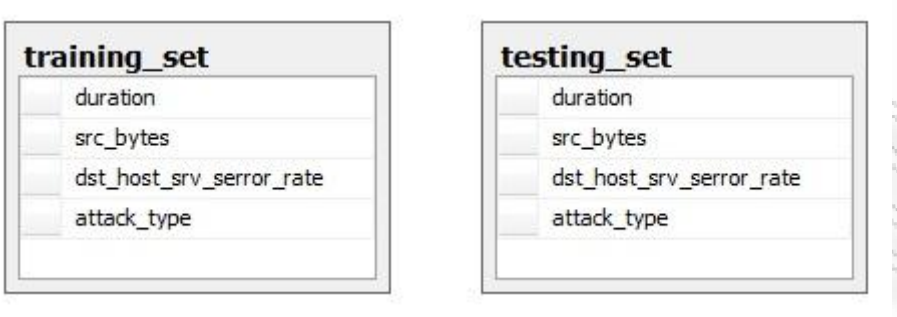
Σχήμα 20: Η κλάση DAL

Τέλος, η κλάση Logger (Σχήμα 21), είναι μια στατική κλάση η οποία χρησιμοποιείται για την καταγραφή των συμβάντων κατά τη διάρκεια εκτέλεσης του προγράμματος. Το αρχείο στο οποίο γίνεται η καταγραφή είναι το “gEPLog.txt”. Όπως φαίνεται και στο σχήμα της κλάσης, η καταγραφή των συμβάντων γίνεται σε 3 επίπεδα: Επίπεδο Πληροφορίας (Info), επίπεδο προειδοποιητικό (warning) και επίπεδο λάθους (error). Υπάρχει και άλλη μια μέθοδος για καταγραφή επιπέδου αποσφαλμάτωσης (debug) μόνο για την περίπτωση που γίνονται αλλαγές, επεκτάσεις και διορθώσεις στο πρόγραμμα του αλγορίθμου.



Σχήμα 21: Η κλάση Logger

Όσον αφορά την υλοποίηση της βάσης δεδομένων, αυτή περιέχει δύο πίνακες που αφορούν την εκτέλεση και την αξιολόγηση του αλγορίθμου: Τον πίνακα training_set όπου υπάρχουν τα δεδομένα με τα οποία εκπαιδεύεται ο Γενετικός Αλγόριθμος και ο πίνακας testing_set όπου υπάρχουν τα δεδομένα με τα οποία αξιολογείται τα ποσοστά επιτυχίας του Γενετικού Αλγόριθμου.



Σχήμα 22: Οι πίνακες training_set και testing_set της βάσης δεδομένων

6. Μετρήσεις – Αποτελέσματα

Στην ενότητα αυτή καταγράφεται η διαδικασία και οι παράμετροι που χρησιμοποιήθηκαν στην εξόρυξη δικτυακών εισβολών και τα πειραματικά αποτελέσματα για την αξιολόγηση του Γενετικού Αλγορίθμου στην εξόρυξη δικτυακών εισβολών.

6.1. Δεδομένα εκπαίδευσης και αξιολόγησης

Ως βάση για την δημιουργία του συνόλου δεδομένων για την εκπαίδευση και αξιολόγηση είναι το υποσύνολο που περιέχει το σύνολο δεδομένων KDDCup και συγκεκριμένα το αρχείο: "kddcup.data_10_percent.txt". Το αρχείο αυτό περιέχει το 10% των συνολικών δικτυακών συμβάντων. Το σύνολο αυτό περιέχει 494021 εγγραφές – συνδέσεις, από τις οποίες 97278 είναι κανονικές συνδέσεις, δηλαδή 19,7% των συνολικών συνδέσεων και οι υπόλοιπες 396743 (80,3%) αποτελούν επιθέσεις.

Το παραπάνω γεγονός, δείχνει ότι η κατανομή κανονικών συνδέσεων με επιθέσεις είναι μη ρεαλιστικό [60], καθώς στον πραγματικό κόσμο συμβαίνει ουσιαστικά το αντίθετο, καθώς οι κανονικές συνδέσεις σε ένα δίκτυο υπερτερούν σε σημαντικά μεγάλο βαθμό των εισβολών. Αυτό εισάγει μια αρνητική παράμετρο στην αναζήτηση εισβολών από ένα σύστημα όπως το προτεινόμενο.

Από το παραπάνω υποσύνολο δεδομένων KDD99, επιλέχθηκαν με τυχαίο τρόπο 1000 συνδέσεις οι οποίες αποθηκεύτηκαν στον πίνακα training_set και χρησιμοποιήθηκαν για εκπαίδευση του Γενετικού Αλγορίθμου. Το παραπάνω μεγαλύτερο υποσύνολο, αποθηκεύτηκε στον πίνακα testing_set και χρησιμοποιήθηκε στη δεύτερη φάση για αξιολόγηση των κανόνων εισβολής. Το υποσύνολο εκπαίδευσης περιέχει 800 κανονικές συνδέσεις ενώ οι υπόλοιπες 200 αποτελούσαν επιθέσεις οι οποίες ήταν μέσα στα εξής είδη επιθέσεων: smurf, neptune και portsweep. Οι επιθέσεις smurf και neptune είναι επιθέσεις άρνησης εξυπηρέτησης (Denial of Service attacks), ενώ η portsweep σαρώνει τις δικτυακές πόρτες (ports) ενός υπολογιστικού συστήματος για την ανακάλυψη των υπηρεσιών που εκτελούνται στο σύστημα.

Το υποσύνολο εκπαίδευσης περιείχε 130 επιθέσεις smurf, 45 επιθέσεις neptune και 25 επιθέσεις portsweep. Παρακάτω παρατίθεται υπόδειγμα SQL εντολής για εισαγωγή τυχαίων δεδομένων στον πίνακα εκπαίδευσης training_set:

```
insert into training_set
select top 800 * from testing_set
where attack_type = 0
order by NEWID()

insert into training_set
select top 130 * from testing_set
where attack_type = 1
order by NEWID()

insert into training_set
select top 45 * from testing_set
where attack_type = 2
order by NEWID()

insert into training_set
select top 25 * from testing_set
where attack_type = 6
order by NEWID()
```

όπου 0, 1, 2, 6 για το χαρακτηριστικό attack_type αντιπροσωπεύει τις κανονικές, smurf, neptune, portsweep επιθέσεις αντίστοιχα.

6.2. Πειραματικά αποτελέσματα

6.2.1. Παράμετροι εκπαίδευσης

Για την εκπαίδευση του αλγορίθμου στην εξόρυξη εισβολών χρησιμοποιήθηκαν διάφοροι συνδυασμοί γενετικών παραμέτρων. Η μέγιστη γενιά που ορίστηκε ήταν 300 γενιές ενώ ο πληθυσμός των χρωμοσωμάτων που δοκιμάστηκε ήταν 100 και 200 χρωμοσώματα. Το μέγεθος επώασης (Brood size), δοκιμάστηκε τόσο με 1 όσο και με 2, δηλαδή στην δεύτερη περίπτωση ο γονέας παράγει 2 παιδιά τα οποία επιδέχονται γενετικές αλλαγές. Για την διασταύρωση δοκιμάστηκε μόνον η διασταύρωση ενός σημείου, εξάλλου τα γονίδια που συνιστούσαν το χρωμόσωμα ήταν 3, οπότε η διασταύρωση ενός σημείου θεωρείται επαρκής για το πρόβλημα αυτό. Η επιλογή στον συγκεκριμένο αλγόριθμο έχει σημασία καθώς υλοποιούνται 2 είδη επιλογών: η επιλογή γονέα για την παραγωγή απογόνων και η επιλογή επιβίωσης των απογόνων. Σε αυτή την περίπτωση χρησιμοποιήθηκε η λογική της «πίεσης επιλογής» [58], κατά την οποία ανάλογα με το συνδυασμό των παραμέτρων επιλογής είναι αδύναμη ή δυνατή. Παρακάτω αναφέρονται με τη σειρά αδύναμης προς δυνατής οι γενετικοί παράμετροι επιλογής που υλοποιήθηκαν στον αλγόριθμο:

- Ομοιόμορφη στοχαστική (Uniform stochastic)
- Επιλογή ρουλέτας (Fitness proportional)
- Δυαδικού τουρνουά (Binary tournament)

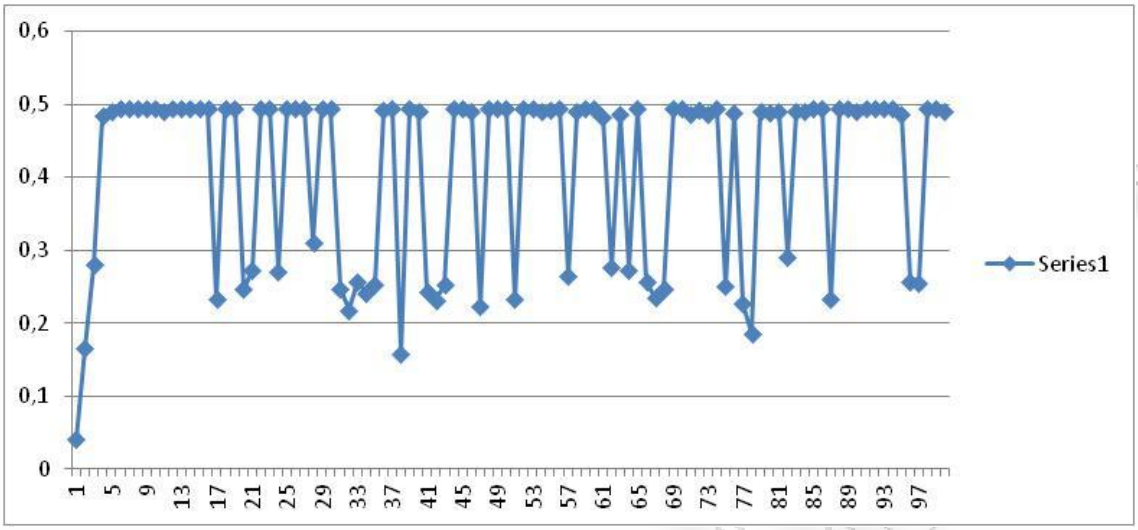
Η επίδραση των παραπάνω παραγόντων στον αλγόριθμο έχει ως αντίκτυπο τον έλεγχο της εστίασης στην αναζήτηση στις επόμενες γενιές. Συνδυασμός υπερβολικά υψηλής πίεσης επιλογής έχει ως αποτέλεσμα την πιθανή σύγκλιση του αλγορίθμου σε τοπικά μέγιστα στο πεδίο αναζήτησης. Ως συνέπεια, ένας από τους δύο μηχανισμούς επιλογής συνήθως είναι ομοιόμορφης επιλογής.

Στον προτεινόμενο αλγόριθμο, χρησιμοποιήθηκαν συνδυασμοί αυτών με τα καλύτερα αποτελέσματα να παράγονται από συνδυασμό επιλογής με ομοιόμορφη επιλογή.

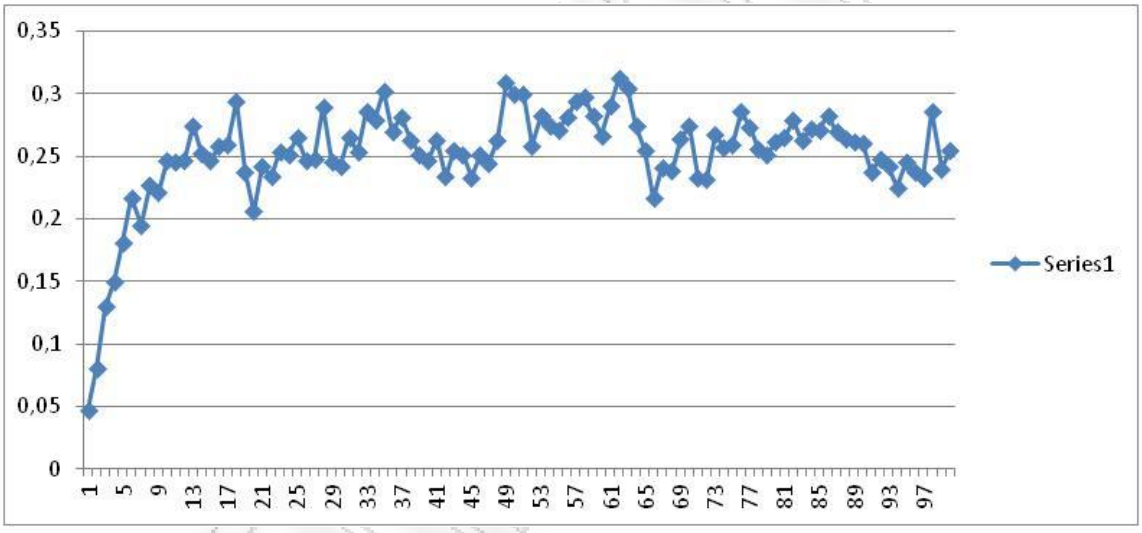
Όσον αφορά την μετάλλαξη, χρησιμοποιήθηκε πιθανότητα 0,2 και 0,33 δηλαδή στατιστικά ένα γονίδιο από κάθε χρωμόσωμα να επιδέχεται μετάλλαξη. Η διαφορά αυτή στη μετάλλαξη δεν είχε ιδιαίτερη επίπτωση στον αλγόριθμο. Η μετάλλαξη που υλοποιήθηκε και χρησιμοποιήθηκε, ήταν η τυχαία, κατά την οποία η τιμή του γονιδίου που επιδέχεται μετάλλαξη, αλλάζει μέσα από το δυνατό πλήθος τιμών που υπάρχει στο συλλογή δεδομένων εκπαίδευσης.

Στο τέλος της διαδικασίας, τα καλύτερα χρωμοσώματα ήταν η λίστα η οποία παράγει ο αλγόριθμος και χρησιμοποιήθηκαν για την αξιολόγηση της επίδοσης του αλγορίθμου.

Η πίεση επιλογής στον συγκεκριμένο αλγόριθμο φαίνεται εν μέρει από τα παρακάτω διαγράμματα στα οποία το πρώτο έχει υλοποιηθεί με επιλογή δυαδικού τουρνουά για την επιλογή γονέα και εξαναγκασμένης ρουλέτας για την επιλογή επιβίωσης ενώ στο δεύτερο με επιλογή δυαδικού τουρνουά για την επιλογή γονέα και ομοιόμορφης στοχαστικής για την επιλογή επιβίωσης. Στα διαγράμματα καταγράφεται η μέση τιμή της απόδοσης όλων των χρωμοσωμάτων για τις πρώτες 100 γενιές του αλγορίθμου. Παρόλη την μικρότερη μέση τιμή απόδοσης που έχουν τα χρωμοσώματα του δεύτερου πειράματος, οδηγούν σε καλύτερα αποτελέσματα στο τέλος του αλγορίθμου.



Σχήμα 23: Απεικόνιση μέσης απόδοσης σε πείραμα το οποίο συνδυάζει επιλογή τουρνουά και εξαναγκασμένης ρουλέτας



Σχήμα 24: Απεικόνιση μέσης απόδοσης σε πείραμα το οποίο συνδυάζει επιλογή τουρνουά και ομοιόμορφης επιλογής

6.2.2. Αποτελέσματα

Τα καλύτερα αποτελέσματα του αλγορίθμου είχαν τις παρακάτω παραμέτρους:

- Μέγεθος πληθυσμού: 100
- Μέγεθος επώασης: 2
- Μέγιστη γενιά: 300
- Διασταύρωση: Ενός σημείου με πιθανότητα 0,6
- Μετάλλαξη: Τυχαία με πιθανότητα 0,2
- Επιλογή γονέων: Εξαναγκασμένη ρουλέτα
- Επιλογή επιβίωσης: Ομοιόμορφη στοχαστική

Το αποτέλεσμα του συνόλου των καλύτερων χρωμοσωμάτων – εισβολών παρατίθεται παρακάτω (Πίνακας 10):

Απόδοση	duration	src_bytes	dst_host_srv_serror_rate	γενιά που παράχθηκε
0,495	0	1032	0	1
0,155	0	520	0	1
0,0875	0	0	0	1
0,19	0	0	100	4
0,005	1	0	0	11
0,005	24	0	0	121

Πίνακας 10: Σύνολο από εισβολές που εξορύχτηκαν από την εκπαίδευση του αλγορίθμου

Αντίστοιχα αποτελέσματα υπήρχαν, όταν στην επιλογή γονέων υπήρχε και η δυαδική επιλογή τουρνουά και στην επιλογή επιβίωσης παραμένει η ομοιόμορφη. Επίσης, ο αλγόριθμος αρκεί να εκτελεστεί με λιγότερες γενιές για να παραχθούν τα παραπάνω αποτελέσματα (έως 200 γενιές).

Οι τελικοί κανόνες που εξορύχτηκαν μπορούν να ελεγχθούν για την ακρίβειά τους τόσο στο σύνολο εκπαίδευσης όσο και στο σύνολο αξιολόγησης με το παρακάτω ερώτημα στη βάση δεδομένων:

```
select COUNT(*) from training_set where attack_type != 0 and
((duration=0 and src_bytes=1032 and dst_host_srv_serror_rate=0) or
 (duration=0 and src_bytes=520 and dst_host_srv_serror_rate=0) or
 (duration=0 and src_bytes=0 and dst_host_srv_serror_rate=0) or
 (duration=0 and src_bytes=0 and dst_host_srv_serror_rate=100) or
 (duration=1 and src_bytes=0 and dst_host_srv_serror_rate=0) or
 (duration=24 and src_bytes=0 and dst_host_srv_serror_rate=0))
```

και:

```
select COUNT(*) from testing_set where attack_type != 0 and
((duration=0 and src_bytes=1032 and dst_host_srv_serror_rate=0) or
 (duration=0 and src_bytes=520 and dst_host_srv_serror_rate=0) or
 (duration=0 and src_bytes=0 and dst_host_srv_serror_rate=0) or
 (duration=0 and src_bytes=0 and dst_host_srv_serror_rate=100) or
 (duration=1 and src_bytes=0 and dst_host_srv_serror_rate=0) or
 (duration=24 and src_bytes=0 and dst_host_srv_serror_rate=0))
```

Τα αποτελέσματα συνοψίζονται στον παρακάτω πίνακα:

Σύνολο δεδομένων εκπαίδευσης				**Σύνολο δεδομένων αξιολόγησης**			
Detection rate		**False positive**		**Detection rate**		**False positive**	
πλήθος	ποσ. (%)	πλήθος	ποσ. (%)	πλήθος	ποσ. (%)	πλήθος	ποσ. (%)
198	99%	42	5,3%	390443	98,4%	5331	5,5%

Πίνακας 11: Αποτελέσματα ανίχνευσης εισβολών

Από τα αποτελέσματα, φαίνεται το πολύ καλό ποσοστό ανίχνευσης εισβολών παρόλη τη χρήση ενός μικρού υποσυνόλου δεδομένων (1000 εγγραφές). Τα ποσοστά επιτυχίας στο μεγαλύτερο σύνολο δεδομένων (494021 εγγραφές) παραμένουν σε υψηλά επίπεδα. Το ποσοστό false positive στο σύνολο δεδομένων εκπαίδευσης έχει ένα ποσοστό 5,3% ενώ σε αντίστοιχα επίπεδα είναι και στο σύνολο δεδομένων αξιολόγησης (5,5%). Αυτό συμβαίνει καθώς υπάρχουν συνδυασμοί χαρακτηριστικών που ισχύουν τόσο σε κανονική σύνδεση όσο και σε εισβολή. Παράδειγμα στο σύνολο εκπαίδευσης, ο κανόνας `(duration=0 and src_bytes=0 and dst_host_srv_serror_rate=0)`, έχει 28 εγγραφές που καταγράφεται ως εισβολή και 42 εγγραφές που καταγράφεται ως κανονική. Ως συνέπεια, ο συγκεκριμένος κανόνας εισβολής αυξάνει το δείκτη false positive.

Αν εξαιρέσουμε τον κανόνα αυτόν, το ποσοστό ανίχνευσης μειώνεται, μηδενίζεται όμως το ποσοστό λανθασμένων συναγερμών. Ανάλογα με τη πολιτική προστασίας του δικτυακού περιβάλλοντος, υπάρχει η δυνατότητα για ρύθμιση αυτών των παραγόντων.

Σύνολο δεδομένων εκπαίδευσης				**Σύνολο δεδομένων αξιολόγησης**			
Detection rate		**False positive**		**Detection rate**		**False positive**	
πλήθος	ποσ. (%)	πλήθος	ποσ. (%)	πλήθος	ποσ. (%)	πλήθος	ποσ. (%)
170	85%	0	0	367804	92,7%	13	0

Πίνακας 12: Αποτελέσματα ανίχνευσης εισβολών με έλεγχο των κανόνων

Ένα μειονέκτημα στην επιλογή μικρού πλήθους χαρακτηριστικών, όπως φαίνεται παραπάνω, είναι η πιθανότητα επικάλυψης εισβολών και κανονικών συνδέσεων, να υπάρχουν δηλαδή για τον ίδιο συνδυασμό δικτυακών χαρακτηριστικών, τόσο εισβολές όσο και κανονικές συνδέσεις.

Παρόλα αυτά, με την επιλογή μικρού πλήθους δικτυακών χαρακτηριστικών η διαδικασία εκπαίδευσης γίνεται γρηγορότερα ενώ ταυτόχρονα διατηρείται υψηλό ποσοστό ανίχνευσης. Επίσης ο Αλγόριθμος παρουσιάζει προσαρμοστική τάση, καθώς διατηρεί υψηλά επίπεδα ανίχνευσης στο σύνολο δεδομένων αξιολόγησης, το οποίο αποτελεί ένα σύνολο δεδομένων περίπου 500 φορές μεγαλύτερο από το σύνολο δεδομένων εκπαίδευσης.

Επίσης, τα μειονεκτήματα του υποσύνολου δεδομένων KDD99 [61] όπως, η εισαγωγή νέων επιθέσεων στο υποσύνολο δεδομένων εκπαίδευσης οι οποίες είναι παρόμοιες με τις κανονικές συνδέσεις, οδηγούν σε πιθανή λάθος ταξινόμηση του είδους της σύνδεσης.

Στην προσέγγιση αυτή, παρόλα αυτά, υπάρχει το πλεονέκτημα της γρήγορης εκπαίδευσης για νέου είδους εισβολές. Αυτό σημαίνει πως σε ένα πραγματικό περιβάλλον υπάρχει δυνατότητα σε τακτά χρονικά διαστήματα, να εκπαιδεύεται εκ νέου το σύστημα και να προστίθενται ή να ανανεώνονται έτσι, νέοι κανόνες εισβολής.

7. Ανακεφαλαίωση – Περαιτέρω έρευνα

7.1. Ανακεφαλαίωση

Στην εργασία αυτή παρουσιάστηκε μια προσέγγιση ενός Γενετικού Αλγόριθμου στην ανίχνευση εισβολών και η υλοποίησή του σε λογισμικό. Ο Γενετικός αλγόριθμος εκπαιδεύτηκε σε ορισμένα δεδομένα καταγραφής του δικτύου ώστε να παράγει ένα σύνολο κανόνων εισβολής. Οι κανόνες που εξορύχτηκαν, εφαρμόστηκαν για να ταξινομήσουν την κίνηση ενός υποτιθέμενου πραγματικού περιβάλλοντος, σε φυσιολογική ή εισβολή. Για την εκπαίδευση και ανίχνευση χρησιμοποιήθηκε η συλλογή δεδομένων KDD99Cup. Το πλήθος των δικτυακών χαρακτηριστικών που χρησιμοποιήθηκε για την εκπαίδευση και την ανίχνευση εισβολών, ήταν μικρό ώστε η ανίχνευση να γίνεται γρήγορα και η εκπαίδευση του αλγορίθμου για εξαγωγή νέων κανόνων να είναι εφικτή σε πραγματικό περιβάλλον. Τα αποτελέσματα της προσέγγισης, βασιζόμενα στην συλλογή δεδομένων KDD99, ήταν αποδεκτά, διατηρώντας σε υψηλά επίπεδα την ανίχνευση επιθέσεων, ενώ παράλληλα ο χρόνος εκπαίδευσης ήταν μικρός.

7.2. Περαιτέρω έρευνα

Ο διαχωρισμός φυσιολογικής κίνησης από επιθέσεις είναι το πιο σημαντικό στοιχείο για ένα σύστημα ανίχνευσης εισβολών σε ένα δίκτυο. Εν τούτοις, η ταξινόμηση των απειλών είναι σημαντικός παράγοντας στην ανάλυση των εισβολών σε ένα δίκτυο ώστε να καθοριστούν περαιτέρω οι πολιτικές ασφάλειας του δικτύου και αποτελεί αναπόσπαστο κομμάτι σε ένα μοντέρνο σύστημα ανίχνευσης εισβολών. Η προτεινόμενη προσέγγιση μπορεί να εμπλουτιστεί ώστε να περιλαμβάνει και ανίχνευση συγκεκριμένου τύπου απειλών. Σε αυτό μπορεί να βοηθήσει η προσθήκη μιας επιπλέον αντικειμενικής συνάρτησης και ενός σταδίου, αυτή του πλαισίου υποστήριξης – σιγουριάς (support – confidence) [51] διατηρώντας την φαινοτυπική αναπαράσταση του αλγορίθμου και τα δικτυακά χαρακτηριστικά. Παρόλα αυτά, για να διατηρηθεί η ταχύτητα εκπαίδευσης θα πρέπει να αναζητηθούν και άλλες λύσεις.

Επίσης, υπάρχει δυνατότητα για έρευνα με επιπλέον τεχνικές μείωσης διαστάσεων και να αξιολογηθούν με βάση την προτεινόμενη υλοποίηση. Επιπρόσθετα, η υλοποίηση του Γενετικού αλγόριθμου θα μπορούσε να εμπλουτιστεί με επιπλέον γενετικές παραμέτρους ή να αξιοποιηθεί η δυνατότητα επικαλυπτόμενων γενεών (Overlapping generations), ώστε τα καλύτερα χρωμοσώματα όχι μόνο να εξάγονται, αλλά και να χρησιμοποιούνται στην αναπαραγωγή των επόμενων γενεών.

Τέλος τα χρωμοσώματα θα μπορούσαν να εμπλουτιστούν με επιπλέον αναπαραστάσεις στα γονίδιά τους ώστε να πιάνουν διαστήματα τιμών των δικτυακών χαρακτηριστικών. Παράδειγμα θα μπορούσε να προστεθεί στο πεδίο τιμών ενός γονιδίου π.χ. του src_bytes και ο αριθμός -1, το οποία θα σήμαινε «οτιδήποτε». Επίσης θα μπορούσαν να χρησιμοποιηθούν και οι τελεστές σύγκρισης στην εκπαίδευση του αλγορίθμου. Αυτό αναμένεται να συνεισφέρει στην προσαρμοστικότητα του αλγορίθμου σε νέου είδους επιθέσεις.

Βιβλιογραφία

1. KDD Cup 1999: data. Available at: http://www.sigkdd.org/kddcup/index.php?section=1999&method=data
2. Banković, Z., Stepanović, D., Bojanić, S., Nieto-Taladriz, O.: Improving network security using genetic algorithm approach. Computers and Electrical Engineering 33(5 - 6), 438-451 (September 2007)
3. Kotzanikolaou, P., Douligeris, C.: Computer Network Security: Basic Background and Current Issues. In : Network Security: Current Status and Future Directions. John Wiley and Sons (2007) 1-12
4. Menezes, A., Oorschot, P., Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton, FL (1997)
5. Bace, R., Mell, P.: NIST Special Publication on Intrusion Detection Systems., National Institute of Standards and Technology (2001)
6. Anderson, J. P.: Computer Security Threat Monitoring and Surveillance. Technical report, James P Anderson Co., Fort Washington, Pennsylvania (1980)
7. Γκρίτζαλης, Σ., Κάτσικας, Σ., Γκρίτζαλης, Δ.: Ασφάλεια δικτύων υπολογιστών: τεχνολογίες και υπηρεσίες σε περιβαλλοντα ηλεκτρονικού επιχειρείν και ηλεκτρονικής διακυβέρνησης. Παπασωτηρίου, Αθήνα (2003)
8. Heady, R., Luger, G., Maccabe, A., Servilla, M.: The Architecture of a Network Level Intrusion Detection System. Technical Report CS90-20, University of New Mexico (1990)
9. Richardson, R.: 2008 CSI Computer Crime & Security Survey., Computer Security Institute, San Francisco (2008)
10. Gordon, L., Loeb, M., Lucyshyn, W., Richardson, R.: 2006 CSI/FBI Computer Crime and Security Survey., Computer Security Institute, San Francisco (2006)
11. Gordon, L., Loeb, M., Lucyshyn, W., Richardson, R.: 2005 CSI/FBI Computer Crime and Security Survey., Computer Security Institute, San Francisco (2005)
12. Gordon, L., Loeb, M., Lucyshyn, W., Richardson, R.: 2004 CSI/FBI Computer Crime and Security Survey., Computer Security Institute, San Francisco (2004)
13. McEachen, J., Zachary, J.: IDS for Networks. In Douligeris, C., Serpanos, D., eds. : Network Security: Current Status and Future Directions. John Wiley and Sons (2007)
14. Moore, D., Shannon, C., Voelker, G., Savage, S.: Internet quarantine: Requirements for containing self propagating code. In : Proceedings of the IEEE INFOCOM, San Francisco, vol. III, pp.1901-1910 (2003)
15. Escamilla, T.: Intrusion detection: Network security beyond the firewall. John Wiley & Sons, Inc. (1998)
16. Anderson, J.: Computer security threat monitoring and surveillance. Technical Report 98-17, James P Anderson Co., FortWashington, Pennsylvania (1980)
17. Mandia, K., Prosise, C., Pepe, M.: Incident Response and Computer Forensics. McGraw Hill/Osborne, Emeryville, CA (2003)
18. Li, H., Chang, L., Wang, X.: A Useful Intrusion Detection System Prototype to Monitor Multi-processes Based on System Calls. In Qing, S., Okamoto, T., Zhou, J., eds. : Information and Communications Security 2229. Springer Berlin/Heidelberg (2001) 441-450
19. Hofmeyr, S.: An Immunological Model of Distributed Detection and Its Application to

Computer Security. PhD Thesis, University of New Mexico (1999)
20. Denning, D.: An Intrusion-Detection Model. IEEE Transactions on Software Engineering SE-13(2), 222-232 (1987)
21. Patcha, A., Park, J.-M.: An overview of anomaly detection techniques: Existing solutions and latest technological trends. Computer Networks 51(12), 3448-3470 (2007)
22. Roesch, M.: Snort - Lightweight intrusion detection for networks. In : LISA '99: Proceedings of the 13th USENIX conference on System administration, Berkeley, CA, USA, pp.229-238 (1999)
23. Μητροκώτσα, Α.: Ανίχνευση εισβολών σε δίκτυα υπολογιστών με αλγόριθμους μηχανικής μάθησης. PhD Thesis, Πανεπιστήμιο Πειραιώς, Πειραιάς (2007)
24. Ko, C., Ruschitzka, M., Levitt, K.: Execution monitoring of security-critical programs in distributed systems: a Specification-based approach. In : Proceedings of 1997 IEEE Symposium on Security and Privacy, pp.175-187 (1997)
25. Sekar, R., Gupta, A., Frullo, J., Shanbhag, T., Tiwari, A., Yang, H., Zhou, S.: Specification-based anomaly detection: a new approach for detecting network intrusions. In : Proceedings of the 9th ACM conference on Computer and communications security, Washington, DC, pp.265-274 (2002)
26. Axelsson, S.: Research in intrusion detection systems: A survey. Technical Report, Chalmers University of Technology, Goteberg, Sweden (1999)
27. Mitchell, M.: An Introduction to Genetic Algorithms 5th edn. MIT Press (1999)
28. Λυκοθανάσης, Σ.: Γενετικοί Αλγόριθμοι και Εφαρμογές. Ελληνικό Ανοικτό Πανεπιστήμιο, Πάτρα (2001)
29. Holland, J.: Adaptation in Natural and Artificial Systems 21992nd edn. University of Michigan Press (1975)
30. Goldberg, D.: Genetic Algorithms in Search, Optimization, and Machine Learning. Addison−Wesley (1989)
31. Optimization Methods: Genetic Algorithms. In: Virtual Institute of Applied Science. Available at: http://www.vias.org/tmdatanaleng/cc_optim_meth_combi.html
32. Jackson, A.: Genetic Algorithms for Use in Financial Problems. AFIR Colloquium, Australia (1997)
33. Caruana, R. A., Schaffer, J. D.: Representation and hidden bias: Gray vs. binary coding for genetic algorithms. In : Proceedings of the Fifth International Conference on Machine Learning (1988)
34. Baker, J.: Adaptive selection methods for genetic algorithms. In : Proceedings of the First International Conference on Genetic Algorithms and Their Applications, pp.101-111 (1985)
35. Goldberg, D., Deb, K.: A comparitive analysis of selection schemes used in genetic algorithms. In : Foundations of genetic algorithms. Morgan Kaufmann (1991) 69-93
36. Mühlenbein, H.: How genetic algorithms really work: 1. Mutation and hillclimbing. In : Parallel Problem Solving from Nature. (1992)
37. Spears, W. M.: Crossover or mutation? In : Proceedings of Foundations of Genetic Algorithms Workshop, pp.221-237 (1992)
38. Haupt, R., Haupt, S.: Practical Genetic Algorithms 2nd edn. John Willey & Sons, New Jersey (2004)
39. Jong, K.: Analysis of the behavior of a class of genetic adaptive systems. Ph.D. Dissertation,

University of Michigan, 1975 (1975)

40. Grefenstette, J.: Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man and Cybernetics 16(1), 122-128 (1986)
41. Eiben, A., Hinterding, R., Michalewicz, Z.: Parameter control in evolutionary algorithms. IEEE Transactions on Evolutionary Computation 3(2), 124-141 (1999)
42. Crosbie, M., Spafford, E.: Applying Genetic Programming to Intrusion Detection. In : Working Notes for the AAAI Symposium on Genetic Programming, MIT, Cambridge, MA, USA, pp.1-8 (1995)
43. Bridges, S., Vaughn, R.: Fuzzy Data Mining And Genetic Algorithms Applied To Intrusion Detection. In : Proceedings of 12th Annual Canadian Information Technology Security Symposium, pp.109-122 (2000)
44. Lu, W., Traore, I.: Detecting New Forms of Network Intrusion Using Genetic Programming., 475-494 (2004)
45. Middlemiss, M., Dick, G.: Feature selection of intrusion detection data using a hybrid genetic algorithm/KNN approach. In : Design and application of hybrid intelligent systems. IOS Press, Amsterdam (2003) 519-527
46. Xia, T., Qu, G., Hariri, S., Yousif, M.: An efficient network intrusion detection method based on information theory and genetic algorithm. In : Performance, Computing, and Communications Conference, 2005. IPCCC 2005. 24th IEEE International, pp.11-17 (2005)
47. Chittur, A.: Model generation for an intrusion detection system using genetic algorithm. High School Honors Thesis, Ossining High School, NY (2001)
48. Koza, J.: Genetic programming: On the programming of computers by means of natural selection. MIT Press (1992)
49. Li, W.: A Genetic Algorithm Approach to Network Intrusion Detection. GSEC Practical Assignment, SANS Institute (2003)
50. Pillai, M., Eloff, J., Venter, H.: An approach to implement a network intrusion detection system using genetic algorithms. In : Proceedings of the SAICSIT, pp.221-228 (2004)
51. Gong, R., Zulkernine, M., Abolmaesumi, P.: A software implementation of a genetic algorithm based approach to network intrusion detection. In : Proceedings of the Sixth International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing and First ACIS International Workshop on Self-Assembling Wireless Networks (SNPD/SAWN'05), pp.246-253 (2005)
52. Dasgupta, D., Gonzalez, F.: An Intelligent Decision Support System for Intrusion Detection and Response. In : Information Assurance in Computer Networks. Springer Berlin / Heidelberg (2001)
53. Bankovic, Z., Moya, J., Araujo, Á., Bojanic, S., Nieto-Taladriz, O.: A Genetic Algorithm-based Solution for Intrusion Detection., 192-199 (2009)
54. Hettich, S., Bay, S.: The UCI KDD Archive. Available at: http://kdd.ics.uci.edu
55. Lippmann, R., Haines, J., Fried, D., Korba, J., Das, K.: The 1999 DARPA off-line intrusion detection evaluation. Computer Networks 34(4), 579-595 (2000)
56. Jolliffe, I.: Principal component analysis. Springer (2002 2nd ed.)
57. Mardia, K., Kent, J., Bibby, J.: Multivariate Analysis (Probability and Mathematical Statistics). Academic Press (1995)
58. De Jong, K.: Evolutionary Computation. A Unified Approach. The MIT Press (2006)

59. Static Classes and Static Class Members (C# Programming Guide). In: msdn. Available at: http://msdn.microsoft.com/en-us/library/79b3xss3(VS.80).aspx
60. McHugh, J.: Testing Intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. ACM Transactions on Information and System Security (TISSEC) 3(4), 262-294 (2000)
61. Bouzida, Y., Cuppens, F.: Detecting Known and Novel Network Intrusions. In : Security and Privacy in Dynamic Environments Volume 201. Springer, Boston (2006) 258-270

Παράρτημα Α: Κώδικας υλοποίησης του αλγορίθμου

Α.1 Η κλάση Gene (Gene.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace gEP
{
  /// <summary>
  /// A class that represents a single gene of a chromosome.
  /// </summary>
  class Gene : ICloneable
  {
    private int value;
    private int lowerBound;
    private int upperBound;

    #region Public Properties

    /// <summary>
    /// Gets or Sets the value of a gene
    /// </summary>
    public int Value
    {
      get { return this.value; }
      set { this.value = value; }
    }

    /// <summary>
    /// Gets the lower possible bound value of a gene
    /// </summary>
    public int LowerBound
    {
      get { return this.lowerBound; }
    }

    /// <summary>
    /// Gets the upper possible bound value of a gene
    /// </summary>
    public int UpperBound
    {
      get { return this.upperBound; }
    }

    #endregion

    /// <summary>
    /// Constructor using only lower and upper bounds.
    /// It is mostly used for creating the configuration
    /// genes as it does not initialize the value of the gene.
    /// </summary>
    /// <param name="lowerBD">The lower possible value of the gene</param>
    /// <param name="upperBD">The maximum possible value of the gene</param>
    public Gene(int? lowerBD, int? upperBD)
    {
      if (lowerBD.HasValue)
        lowerBound = lowerBD.Value;
```

```
      else
        lowerBound = Int32.MinValue;
      if (upperBD.HasValue)
        upperBound = upperBD.Value;
      else
        upperBound = Int32.MaxValue;
    }

    /// <summary>
    /// This is the Default constructor of Gene Class.
    /// </summary>
    /// <param name="rnd">a Random Class Instance</param>
    /// <param name="lowerBD">The lower possible value of the gene</param>
    /// <param name="upperBD">The maximum possible value of the gene</param>
    public Gene(Random rnd, int? lowerBD, int? upperBD)
    {
      if (lowerBD.HasValue)
        lowerBound = lowerBD.Value;
      else
        lowerBound = Int32.MinValue;
      if (upperBD.HasValue)
        upperBound = upperBD.Value;
      else
        upperBound = Int32.MaxValue;
      this.value = rnd.Next(lowerBound, upperBound);
    }

    public bool IsValid(int newValue)
    {
      if (newValue < lowerBound || newValue > upperBound)
        return false;
      else
        return true;
    }

    #region ICloneable Members

    /// <summary>
    /// Clones this gene.
    /// </summary>
    /// <returns>A copy of this gene</returns>
    public object Clone()
    {
      Gene copy = new Gene(this.lowerBound, this.upperBound);
      copy.Value = this.value;
      return copy;
    }

    #endregion
  }
}
```

A.2 Η κλάση Chromosome (Chromosome.cs)

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data.SqlClient;

namespace gEP
{
  /// <summary>
  /// A class that holds information about a chromosome.
  /// It contains a collection of genes.
  /// </summary>
  class Chromosome : List<Gene>, ICloneable
  {
    private int numberOfGenes;
    private double fitness;

    public static int mutationCount = 0;
    public static int crossoverCount = 0;
    public static int invalidMutationCount = 0;

    #region Public Properties

    /// <summary>
    /// Returns the number of genes in the chromosome.
    /// this is the same as the count() method of the base class (List of Gene)
    /// but it stays for better semantics.
    /// </summary>
    public int NumberOfGenes
    {
      get { return this.numberOfGenes; }
    }

    ///// <summary>
    ///// Returns a Gene Collection. A gene located
    ///// at a specified index can be accessed.
    ///// </summary>
    //public List<Gene> Genes
    //{
    //    get { return this.genes; }
    //}

    /// <summary>
    /// Returns or sets the chromosome's fitness.
    /// </summary>
    public double Fitness
    {
      get { return this.fitness; }
      set { this.fitness = value; }
    }

    #endregion

    /// <summary>
    /// Chromosome default constructor.
    /// It does not asign initial values to it's genes.
    /// </summary>
    public Chromosome()
      : base(Configuration.NumberOfGenes)
```

```
    {
      numberOfGenes = Configuration.NumberOfGenes;
      for (int i = 0; i < numberOfGenes; i++)
      {
        this.Add(new Gene(Configuration.ConfigGenes[i].LowerBound,
Configuration.ConfigGenes[i].UpperBound));
      }
    }

    /// <summary>
    /// Chromosome constructor. It uses a random number generator
    /// to assign initial values to each gene and the Configuration
    /// Class in order to specify the lower and upper bounds.
    /// </summary>
    /// <param name="rand"></param>
    public Chromosome(Random rand)
      : base(Configuration.NumberOfGenes)
    {
      numberOfGenes = Configuration.NumberOfGenes;
      for (int i = 0; i < numberOfGenes; i++)
      {
        this.Add(new Gene(rand, Configuration.ConfigGenes[i].LowerBound,
Configuration.ConfigGenes[i].UpperBound));
      }
    }

    public void Initialize(Random rand)
    {
      this[0].Value = (int)Configuration.Duration[rand.Next(0,
Configuration.Duration.Count)];
      this[1].Value = (int)Configuration.Src_bytes[rand.Next(0,
Configuration.Src_bytes.Count)];
      this[2].Value = (int)Configuration.Dst_host_srv_serror_rate[rand.Next(0,
Configuration.Dst_host_srv_serror_rate.Count)];
    }

    /// <summary>
    /// Function that evaluates a chromosome's fitness.
    /// </summary>
    public void Evaluate()
    {
      string selectString;
      // Get values for all genes
      int duration;
      int src_bytes;
      int dst_host_srv_serror_rate;

      duration = this[0].Value;
      src_bytes = this[1].Value;
      dst_host_srv_serror_rate = this[2].Value;

      // prepare where clause strings
      // duration
      string durationString = " duration = " + duration;
      // src_bytes
      string src_bytesString = " and src_bytes = " + src_bytes;

      // dst_host_srv_serror_rate
```

```
      string dst_host_srv_serror_rateString = " and dst_host_srv_serror_rate = "
+ dst_host_srv_serror_rate;

      int a;
      int b;
      //Compute α
      selectString = string.Format("select count(*) from training_set where {0}
{1} {2}", durationString, src_bytesString, dst_host_srv_serror_rateString) + "
and attack_type != 0"; // α
      a = int.Parse(DAL.ExecuteScalar(selectString).ToString());
      //Compute β
      selectString = string.Format("select count(*) from training_set where {0}
{1} {2}", durationString, src_bytesString, dst_host_srv_serror_rateString) + "
and attack_type = 0"; // β
      b = int.Parse(DAL.ExecuteScalar(selectString).ToString());
      //Compute fitness (α/A - β/B)
      this.fitness = a / 200.0 - b / 800.0;
    }

    /// <summary>
    /// For testing purposes only
    /// </summary>
    ///
    /*
    public void Evaluate1()
    {
        // Keep this code for testing purposes
        this.fitness = 0.0;
        for (int i = 0; i < numberOfGenes; i++)
        {
            this.fitness += Math.Pow(this[i].Value, 2);
        }
    }
    */

    public void Crossover(Random rand, Chromosome mate)
    {
      switch (Configuration.CrossoverType)
      {
        case CrossoverType.ONE_POINT:
          CrossoverOnePoint(rand, mate);
          break;
        case CrossoverType.TWO_POINT:
          break;
        case CrossoverType.UNIFORM:
          break;
        default:
          Logger.LogError("Could not determine the crossover type. Unexpected
Error.");
          break;
      }
    }

    private void CrossoverOnePoint(Random rand, Chromosome mate)
    {
      int cutPoint;

      if (this.numberOfGenes == 1) // We cannot perform crossover with just one
gene.
```

```
        return;

      // Decide whether there is going to be a crossover on the chromosome
      if (rand.NextDouble() <= Configuration.CrossoverProbability)
      {
        if (numberOfGenes == 2)
          this[1] = (Gene)mate[1].Clone();
        else
        {
          cutPoint = rand.Next(numberOfGenes - 2); // Cut point between 0 and
number Of Genes
          for (int i = cutPoint + 1; i < numberOfGenes; i++)
            this[i] = (Gene)mate[i].Clone();
          crossoverCount++;
          //for (int i = 0; i < cutPoint; i++)
          //    this[i] = (Gene)mate[i].Clone();
          //The above did not work right
        }
      }
    }

    public void Mutate(Random rand)
    {
      switch (Configuration.MutationType)
      {
        case MutationType.DELTA_MUTATION:
          MutateDelta(rand);
          break;
        case MutationType.GAUSSIAN_MUTATION:
          MutateGaussian(rand);
          break;
        case MutationType.NONE:
          break;
        case MutationType.RANDOM_MUTATION:
          MutateRandom(rand);
          break;
        default:
          Logger.LogError("Could not determine the mutation type. Unexpected
Error.");
          break;
      }
    }

    /// <summary>
    /// Mutate a given Chromosome using delta mutation.
    /// Each gene has a given probability of being mutated
    /// either incrementally or decrementally by a fixed delta.
    /// </summary>
    /// <param name="rand">The random number generator</param>
    private void MutateDelta(Random rand)
    {
      int newValue = 0;

      for (int i = 0; i < numberOfGenes; i++)
      {
        // Decide whether there is gonna be a mutation
        if (rand.NextDouble() <= Configuration.MutationProbability)
        {
```

```
          // there is gonna be a mutation and now it's time to decide if the
delta is gonna be added or subtracted to the gene value
          if (rand.NextDouble() < 0.5F)
            // in this case we add the mutation delta to the current value
            newValue = this[i].Value + Configuration.MutationDelta;
          else
            // in this case we subtract the mutation delta to the current value
            newValue = this[i].Value - Configuration.MutationDelta;

          // check legality of mutation
          if (this[i].IsValid(newValue))
          {
            this[i].Value = newValue;
            mutationCount++;
          }
          else
            // do nothing. only increase the invalid mutation counter
            invalidMutationCount++;
        }
      }
    }

    /// <summary>
    /// Mutate a given Chromosome using gaussian mutation.
    /// </summary>
    /// <param name="rand">The random number generator</param>
    private void MutateGaussian(Random rand)
    {
      int newValue = 0;

      for (int i = 0; i < numberOfGenes; i++)
      {
        // Decide whether there is gonna be a mutation
        if (rand.NextDouble() <= Configuration.MutationProbability)
        {
          // there is gonna be a mutation
          // stepsize = (2/(sqrt(2pi))*sigma
          newValue = this[i].Value + (int)(1.3F * Configuration.MutationDelta *
Configuration.NextGaussian(rand) + 0.5);

          // check legality of mutation
          if (this[i].IsValid(newValue))
          {
            this[i].Value = newValue;
            mutationCount++;
          }
          else
            // do nothing. only increase the invalid mutation counter
            invalidMutationCount++;
        }
      }
    }

    /// <summary>
    /// Mutate a given Chromosome using random mutation.
    /// The mutation takes place by randomly generating a new value
    /// uniformly from the interval (lowerBound, upperBound).
    /// selected
    /// </summary>
```

```
    /// <param name="rand">The random number generator</param>
    private void MutateRandom(Random rand)
    {
      //int newValue;
      if (rand.NextDouble() <= Configuration.MutationProbability)
      {
        // there is gonna be a mutation
        //this[0].Value = int.Parse(DAL.ExecuteScalar("select top 1 duration from
training_set order by NEWID()").ToString());
        this[0].Value = int.Parse(Configuration.Duration[rand.Next(0,
Configuration.Duration.Count - 1)].ToString());
        /*newValue = int.Parse(Configuration.DistinctDuration[rand.Next(0,
Configuration.DistinctDuration.Count - 1)].ToString());
        while (this[0].Value == newValue)
        {
            newValue = int.Parse(Configuration.DistinctDuration[rand.Next(0,
Configuration.DistinctDuration.Count - 1)].ToString());
        }
        this[0].Value = newValue;*/
        mutationCount++;
      }
      if (rand.NextDouble() <= Configuration.MutationProbability)
      {
        // there is gonna be a mutation
        //this[1].Value = int.Parse(DAL.ExecuteScalar("select top 1 src_bytes
from training_set order by NEWID()").ToString());
        this[1].Value = int.Parse(Configuration.Src_bytes[rand.Next(0,
Configuration.Src_bytes.Count - 1)].ToString());
        /*newValue = int.Parse(Configuration.DistinctSrc_bytes[rand.Next(0,
Configuration.DistinctSrc_bytes.Count - 1)].ToString());
        while (this[1].Value == newValue)
        {
            newValue = int.Parse(Configuration.DistinctSrc_bytes[rand.Next(0,
Configuration.DistinctSrc_bytes.Count - 1)].ToString());
        }
        this[1].Value = newValue;*/
        mutationCount++;
      }
      if (rand.NextDouble() <= Configuration.MutationProbability)
      {
        // there is gonna be a mutation
        //this[2].Value = int.Parse(DAL.ExecuteScalar("select top 1
dst_host_srv_serror_rate from training_set order by NEWID()").ToString());
        this[2].Value =
int.Parse(Configuration.Dst_host_srv_serror_rate[rand.Next(0,
Configuration.Dst_host_srv_serror_rate.Count - 1)].ToString());
        /*newValue =
int.Parse(Configuration.DistinctDst_host_srv_serror_rate[rand.Next(0,
Configuration.DistinctDst_host_srv_serror_rate.Count - 1)].ToString());
        while (this[2].Value == newValue)
        {
            newValue =
int.Parse(Configuration.DistinctDst_host_srv_serror_rate[rand.Next(0,
Configuration.DistinctDst_host_srv_serror_rate.Count - 1)].ToString());
        }
        this[2].Value = newValue;*/
        mutationCount++;
      }
    }
```

```
    #region ICloneable Members

    /// <summary>
    /// Clones this chromosome returning a new copy.
    /// </summary>
    /// <returns>A copy of this chromosome</returns>
    public object Clone()
    {
      Chromosome copy = new Chromosome();

      copy.Fitness = this.fitness;
      for (int i = 0; i < copy.NumberOfGenes; i++)
      {
        copy[i] = (Gene)this[i].Clone();
      }

      return copy;
    }

    #endregion
  }
}
```

A.3 Η κλάση Population (Population.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace gEP
{
  /// <summary>
  /// A class that holds information about the population of the EA.
  /// It contains a collection of chromosomes.
  /// </summary>
  class Population : List<Chromosome>
  {
    private int populationSize;
    //private List<Chromosome> chromosomes;
    private double maxFitness;
    private double minFitness;
    private double avgFitness;

    private static Chromosome globalMax = null;
    private static Chromosome globalMin = null;

    #region Public Properties

    /// <summary>
    /// Returns the number of chromosomes
    /// contained in the population.
    /// </summary>
    public int PopulationSize
    {
      get { return this.populationSize; }
    }

    ///// <summary>
    ///// Returns a Chromosome Collection. A chromosome
    ///// located at a specified index can be accessed.
    ///// </summary>
    //public List<Chromosome> Chromosomes
    //{
    //    get { return this.chromosomes; }
    //}

    public double MaxFitness
    {
      get { return this.maxFitness; }
    }

    public double MinFitness
    {
      get { return this.minFitness; }
    }

    public double AvgFitness
    {
      get { return this.avgFitness; }
    }

    public Chromosome GlobalMax
    {
```

```
      get { return globalMax; }
    }

    public Chromosome GlobalMin
    {
      get { return globalMin; }
    }

    #endregion

    #region Population Constructors

    /// <summary>
    /// This is the default empty constructor
    /// </summary>
    public Population()
    {
    }

    /// <summary>
    /// Population Constructor according to the population size specified.
    /// Used mostly for generating initial population for offspring
    /// </summary>
    /// <param name="rand">Random object for randomized population
creation</param>
    /// <param name="populationSize">The size of the population to create</param>
    public Population(Random rand, int populationSize)
    {
      this.populationSize = populationSize;
      /* Create initial population */
      for (int i = 0; i < populationSize; i++)
      {
        //Chromosome chromosome = new Chromosome(rand);
        Chromosome chromosome = new Chromosome();
        chromosome.Initialize(rand);
        chromosome.Evaluate();
        Add(chromosome);
      }

      //chromosomes = new List<Chromosome>();
      //for (int i = 0; i < this.populationSize; i++)
      //{
      //    chromosomes.Add(new Chromosome(rand));
      //}
    }

    #endregion

    /// <summary>
    /// Create a random initial population
    /// </summary>
    /// <param name="rand">Random object for randomized population
creation</param>
    public void RunReset(Random rand)
    {
      populationSize = Configuration.PopulationSize;

      maxFitness = double.MinValue;
      minFitness = double.MaxValue;
```

```
      /* Create initial population */
      //chromosomes = new List<Chromosome>();
      for (int i = 0; i < populationSize; i++)
      {
        //chromosomes.Add(new Chromosome(rand));
        //Chromosome chromosome = new Chromosome(rand);
        //changed to below code in order to select a valid initial chromosome
(through DB)
        Chromosome chromosome = new Chromosome();
        chromosome.Initialize(rand);

        chromosome.Evaluate();
        Add(chromosome);

        if (globalMax == null || chromosome.Fitness > globalMax.Fitness)
          globalMax = chromosome;
        if (globalMin == null || chromosome.Fitness < globalMin.Fitness)
          globalMin = chromosome;
        if (chromosome.Fitness > maxFitness)
          maxFitness = chromosome.Fitness;
        if (chromosome.Fitness < minFitness)
          minFitness = chromosome.Fitness;
      }
      // Compute average fitness
      avgFitness = ComputeTotalFitness() / populationSize;
    }

    #region selection methods

    public Population Select(EA ea, int offspringSize)
    {
      switch (Configuration.ParentSelectionType)
      {
        case SelectionType.BINARY_TOURNAMENT:
          return SelectBinaryTournament(ea, offspringSize);
        case SelectionType.FITNESS_PROPORTIONAL:
          return SelectFitnessProportional(ea, offspringSize);
        case SelectionType.RANK_PROPORTIONAL:
          return SelectRankProportional(ea, offspringSize);
        case SelectionType.TRUNCATION:
          return SelectTruncation(ea, offspringSize);
        case SelectionType.UNIFORM_DETERMINISTIC:
          return SelectUniformDeterministic(ea, offspringSize);
        case SelectionType.UNIFORM_STOCHASTIC:
          return SelectUniformStochastic(ea, offspringSize);
        default:
          Logger.LogError("Could not determine the parent selection type.
Unexpected Error.");
          return null;
      }
    }

    public Population SelectBinaryTournament(EA ea, int offspringSize)
    {
      Population offspringPopulation = new Population(ea.EARandom,
offspringSize);
      for (int i = 0; i < offspringPopulation.PopulationSize; i++)
      {
```

```
        offspringPopulation[i] = SelectBinaryTournament(ea.EARandom);
      }
      return offspringPopulation;
    }

    private Chromosome SelectBinaryTournament(Random rand)
    {
      int i = rand.Next(0, this.populationSize);
      int j = rand.Next(0, this.populationSize);
      if (this[i].Fitness > this[j].Fitness)
        return (Chromosome)this[i].Clone();
      else
        return (Chromosome)this[j].Clone();
    }

    /// <summary>
    /// Fitness proportional selecetion
    /// </summary>
    /// <param name="ea">Evolution program object</param>
    /// <param name="offspringSize">size of offspring population</param>
    /// <returns></returns>
    public Population SelectFitnessProportional(EA ea, int offspringSize)
    {
      Population offspringPopulation = new Population(ea.EARandom,
offspringSize);

      for (int i = 0; i < offspringPopulation.PopulationSize; i++)
      {
        offspringPopulation[i] = SelectChromosomeRouletteWheel(ea.EARandom);
      }

      return offspringPopulation;
    }

    /// <summary>
    /// Rank Proportional - Not Implemented in current version
    /// </summary>
    /// <param name="ea"></param>
    /// <param name="offspringSize"></param>
    /// <returns></returns>
    public Population SelectRankProportional(EA ea, int offspringSize)
    {
      Population offspringPopulation = new Population(ea.EARandom,
offspringSize);
      return offspringPopulation;
    }

    /// <summary>
    /// Truncation selection - Not implemented in current version
    /// </summary>
    /// <param name="ea"></param>
    /// <param name="offspringSize"></param>
    /// <returns></returns>
    public Population SelectTruncation(EA ea, int offspringSize)
    {
      Population offspringPopulation = new Population(ea.EARandom,
offspringSize);
      return offspringPopulation;
    }
```

```
    /// <summary>
    /// Uniform deterministic selection - Not implemented in current version
    /// </summary>
    /// <param name="ea"></param>
    /// <param name="offspringSize"></param>
    /// <returns></returns>
    public Population SelectUniformDeterministic(EA ea, int offspringSize)
    {
      Population offspringPopulation = new Population(ea.EARandom,
offspringSize);
      return offspringPopulation;
    }

    public Population SelectUniformStochastic(EA ea, int offspringSize)
    {
      Population offspringPopulation = new Population(ea.EARandom,
offspringSize);
      for (int i = 0; i < offspringPopulation.PopulationSize; i++)
      {
        offspringPopulation[i] = SelectUniformStochastic(ea.EARandom);
      }
      return offspringPopulation;
    }

    private Chromosome SelectChromosomeRouletteWheel(Random rand)
    {
      double totalFitness = ComputeTotalFitness();
      double accumulatedSelectionPropability = 0;
      double pick = rand.NextDouble() * 2.0 - 1.0;
      int i;

      if (totalFitness != 0)
      {
        for (i = 0; (accumulatedSelectionPropability < pick) && (i <
this.populationSize); i++)
        {
          accumulatedSelectionPropability += this[i].Fitness / totalFitness;
        }
      }
      else
      {
        i = rand.Next(1, this.populationSize);
      }

      if (i == 0)
        // This is a special case when pick random value is -1.0
        return (Chromosome)this[0].Clone();
      else
        return (Chromosome)this[i - 1].Clone();
    }

    private void SelectFitnessProportional(Random rand, ref Population newPop)
    {
      for (int i = 0; i < newPop.populationSize; i++)
        newPop[i] = SelectChromosomeRouletteWheel(rand);
    }

    /// <summary>
```

```
    /// Randomly select from this population and generate a new population
    /// </summary>
    /// <param name="rand">The Random object</param>
    /// <param name="newPop">A reference where the new population will be
created</param>
    private void SelectUniformStochastic(Random rand, ref Population newPop)
    {
      for (int i = 0; i < newPop.populationSize; i++)
        newPop[i] = SelectUniformStochastic(rand);
    }

    /// <summary>
    /// Randomly select a chromosome from this population
    /// </summary>
    /// <param name="rand">The Random object</param>
    /// <returns></returns>
    private Chromosome SelectUniformStochastic(Random rand)
    {
      int i;
      i = rand.Next(this.populationSize);
      return (Chromosome)this[i].Clone();
    }

    #endregion

    #region Recombination methods

    /// <summary>
    /// Recombine population works in two steps for each chromosome:
    /// The first step is to select from mating pool a mate that will be used
along with the working chromosome.
    /// The selection method for this step will be determined from parent
selection settings.
    /// The second step is to crossover the two chromosomes using the crossover
configuration settings.
    /// </summary>
    /// <param name="rand"></param>
    /// <param name="matingPool"></param>
    public void Recombine(Random rand, Population matingPool)
    {
      // Apply crossover
      switch (Configuration.MateSelectionType)
      {
        case SelectionType.BINARY_TOURNAMENT:
          break;
        case SelectionType.FITNESS_PROPORTIONAL:
          MateFitnessProportinal(rand, matingPool);
          break;
        case SelectionType.UNIFORM_DETERMINISTIC:
          break;
        case SelectionType.UNIFORM_STOCHASTIC:
          break;
        default:
          Logger.LogError("Could not determine the mate selection type.
Unexpected Error.");
          break;
      }

      // Apply mutation
```

```
      for (int i = 0; i < populationSize; i++)
      {
        this[i].Mutate(rand);
      }

    }

    private void MateFitnessProportinal(Random rand, Population matingPool)
    {
      Chromosome mate;

      for (int i = 0; i < this.populationSize; i++)
      {
        mate = matingPool.SelectChromosomeRouletteWheel(rand);
        this[i].Crossover(rand, mate);
      }
    }

    #endregion

    public void SurvivalSelection(Random rand, ref Population newPop)
    {
      // Apply survival selection
      switch (Configuration.SurvivalSelectionType)
      {
        case SelectionType.UNIFORM_STOCHASTIC:
          SelectUniformStochastic(rand, ref newPop);
          break;
        case SelectionType.FITNESS_PROPORTIONAL:
          SelectFitnessProportional(rand, ref newPop);
          break;
        case SelectionType.TRUNCATION:
          break;
        default:
          Logger.LogError("Could not determine or invalid survival selection
type. Unexpected Error.");
          break;
      }
    }


    public void Evaluate(EA ea)
    {
      maxFitness = double.MinValue;
      minFitness = double.MaxValue;

      for (int i = 0; i < populationSize; i++)
      {
        this[i].Evaluate();

        // Compute local max & min fitness
        if (this[i].Fitness > maxFitness)
          maxFitness = this[i].Fitness;
        if (this[i].Fitness < minFitness)
          minFitness = this[i].Fitness;
        if (this[i].Fitness > 0)
        {
          bool foundChromosome = false;
          for (int k = 0; k < ea.BestPool.Count; k++)
```

```
          {
            bool foundGene = true;
            for (int l = 0; l < Configuration.NumberOfGenes; l++)
            {
              if (ea.BestPool[k][l].Value != this[i][l].Value)
                foundGene = false;
            }
            if (foundGene == true)
              foundChromosome = true;
          }
          // if the current chromosome is not included in the best chromosomes
pool the add it to the pool and to the file
          if (foundChromosome == false)
          {
            ea.BestPool.Add(this[i]);
            System.IO.File.AppendAllText(Configuration.PoolFilename,
this[i].Fitness.ToString());
            for (int j = 0; j < this[i].NumberOfGenes; j++)
            {
              System.IO.File.AppendAllText(Configuration.PoolFilename, "\t" +
this[i][j].Value);
            }
            System.IO.File.AppendAllText(Configuration.PoolFilename, "\t" +
ea.CurrentGeneration);
            System.IO.File.AppendAllText(Configuration.PoolFilename,
Environment.NewLine);
          }
        }
      }
      // Compute average fitness
      avgFitness = ComputeTotalFitness() / populationSize;
    }

    public void Statistics(EA ea)
    {
      for (int i = 0; i < populationSize; i++)
      {
        // Compute new Global chromosomes, if any
        if (this[i].Fitness > GlobalMax.Fitness)
        {
          globalMax = this[i];
          ea.GlobalMaxGeneration = ea.CurrentGeneration;
        }
        if (this[i].Fitness < GlobalMin.Fitness)
          globalMin = this[i];
      }
    }

    public void PopulationReport()
    {
      Console.WriteLine("Current Population fitness: max = " + maxFitness + " |
min = " + minFitness + " | avg = " + avgFitness);
      Console.WriteLine("Chrom  Fitness\tgene values");
      for (int i = 0; i < populationSize; i++)
      {
        Console.Write(String.Format("{0,3}", (i + 1).ToString()));
        Console.Write("\t" + String.Format("{0:0.000000}", this[i].Fitness));
        for (int j = 0; j < this[i].NumberOfGenes; j++)
        {
```

```
          Console.Write("\t" + this[i][j].Value);
        }
        Console.WriteLine();
      }
    }

    /// <summary>
    /// Writes to a file information about the population
    /// </summary>
    public void PopulationReportFile()
    {
      string reportFile = Configuration.ReportFilename;
      System.IO.File.AppendAllText(reportFile, "Current Population fitness: max =
" + maxFitness + " | min = " + minFitness + " | avg = " + avgFitness +
Environment.NewLine);
      System.IO.File.AppendAllText(reportFile, "Chrom  Fitness\tgene values" +
Environment.NewLine);
      for (int i = 0; i < populationSize; i++)
      {
        System.IO.File.AppendAllText(reportFile, String.Format("{0,3}", (i +
1).ToString()));
        System.IO.File.AppendAllText(reportFile, "\t" +
String.Format("{0:0.000000}", this[i].Fitness));
        for (int j = 0; j < this[i].NumberOfGenes; j++)
        {
          System.IO.File.AppendAllText(reportFile, "\t" + this[i][j].Value);
        }
        System.IO.File.AppendAllText(reportFile, Environment.NewLine);
      }
    }

    public double ComputeTotalFitness()
    {
      double totalFitness = 0;
      for (int i = 0; i < this.populationSize; i++)
        totalFitness += this[i].Fitness;
      return totalFitness;
    }
  }
}
```

Α.4 Η κλάση EA (EA.cs)

```
using System;
using System.Collections.Generic;
using System.Text;

namespace gEP
{
  /// <summary>
  /// A class that represents an Evolutionary Algorithm.
  /// </summary>
  class EA
  {
    private Population EAPopulation;
    private int currentGeneration = 0;
    private Random eaRandom;
    private Population offspringPopulation;
    private int offspringPopulationSize;
    private int globalMaxGeneration = 0;

    private List<Chromosome> bestPool = new List<Chromosome>();

    #region Public Properties

    /// <summary>
    /// Gets the random object used for random operations
    /// </summary>
    public Random EARandom
    {
      get { return eaRandom; }
    }

    public int GlobalMaxGeneration
    {
      get { return globalMaxGeneration; }
      set { globalMaxGeneration = value; }
    }

    public int CurrentGeneration
    {
      get { return currentGeneration; }
      set { currentGeneration = value; }
    }

    public List<Chromosome> BestPool
    {
      get { return bestPool; }
      set { bestPool = value; }
    }

    #endregion

    public EA()
    {
    }

    public void Run(Random rand)
    {
      eaRandom = rand;
```

```
      offspringPopulationSize = Configuration.PopulationSize *
Configuration.BroodSize;
      //Logger.LogDebug("Offspring population size:" + offspringPopulationSize);
      /* Initialize Population */
      Logger.LogInfo("Initializing population");
      EAPopulation = new Population();
      EAPopulation.RunReset(rand);
      currentGeneration++;
      //GenerationReport();
      //Print first generation
      GenerationReportFile();
      GenerationStatisticsFile();
      //EAPopulation.PopulationReport();
      EAPopulation.PopulationReportFile();
      Logger.LogInfo("Offspring Selection for generation: " + currentGeneration);
      Logger.LogInfo("Maximum fitness: " + EAPopulation.MaxFitness + " Minimum
fitness: " + EAPopulation.MinFitness + " Average fitness: " +
EAPopulation.AvgFitness);
      //Logger.LogDebug("Maximum fitness: " + EAPopulation.GlobalMax.Fitness + "
Minimum fitness: " + EAPopulation.GlobalMin.Fitness);

      while (currentGeneration < Configuration.MaxGenerations)
      {
        // Parent selection
        offspringPopulation = EAPopulation.Select(this, offspringPopulationSize);
        // Recombine offspring population
        offspringPopulation.Recombine(eaRandom, EAPopulation);
        // Evaluate offspring population
        offspringPopulation.Evaluate(this);
        // Survival selection
        offspringPopulation.SurvivalSelection(eaRandom, ref EAPopulation);
        EAPopulation.Evaluate(this);
        //offspringPopulation.PopulationReportFile();

        EAPopulation.Statistics(this);

        currentGeneration++;
        //GenerationReport();
        //Commented for fast run
        //GenerationReportFile();
        GenerationStatisticsFile();
        Logger.LogInfo("Offspring Selection for generation: " +
currentGeneration);
        Logger.LogInfo("Maximum fitness: " + EAPopulation.MaxFitness + " Minimum
fitness: " + EAPopulation.MinFitness + ". Average fitness: " +
EAPopulation.AvgFitness);
        //EAPopulation.PopulationReport();
        //Commented in order to speed up program running
        //EAPopulation.PopulationReportFile();
      }
      // Print last generation
      GenerationReportFile();
      System.IO.File.AppendAllText(Configuration.ReportFilename, "Global Results:
Maximum fitness: " + EAPopulation.GlobalMax.Fitness + " found at Generation: " +
this.globalMaxGeneration + " Minimum fitness: " +
EAPopulation.GlobalMin.Fitness);
      //Console.WriteLine("Global Results: Maximum fitness: " +
EAPopulation.GlobalMax.Fitness + " found at Generation: " +
```

```
this.globalMaxGeneration + " Minimum fitness: " +
EAPopulation.GlobalMin.Fitness);
      Logger.LogInfo("Global Results: Maximum fitness: " +
EAPopulation.GlobalMax.Fitness + " found at Generation: " +
this.globalMaxGeneration + " Minimum fitness: " +
EAPopulation.GlobalMin.Fitness);
    }

    public void GenerationReport()
    {
      Console.WriteLine();
      Console.WriteLine("Population report after generation " +
(currentGeneration).ToString());
      Console.WriteLine("Global fitness: max = " + EAPopulation.GlobalMax.Fitness
+ ", min = " + EAPopulation.GlobalMin.Fitness);
      Console.WriteLine("Total number of crossovers: " +
Chromosome.crossoverCount);
      Console.WriteLine("Total number of mutations: " +
Chromosome.mutationCount);
      // don't need this line for report - no constraint violation on my
configuration
      //Console.WriteLine("Total number of constraint violations: " +
Chromosome.invalidMutationCount);
    }

    public void GenerationReportFile()
    {
      string reportFile = Configuration.ReportFilename;
      //System.IO.File.AppendAllText(reportFile, "Current Population fitness: max
= " + maxFitness + " | min = " + minFitness + " | avg = " + avgFitness +
Environment.NewLine);
      //System.IO.File.AppendAllText(reportFile, "Chrom  Fitness\tgene values" +
Environment.NewLine);
      System.IO.File.AppendAllText(reportFile, Environment.NewLine + "Population
report after generation " + (currentGeneration).ToString() +
Environment.NewLine);
      System.IO.File.AppendAllText(reportFile, "Global fitness: max = " +
EAPopulation.GlobalMax.Fitness + ", min = " + EAPopulation.GlobalMin.Fitness +
Environment.NewLine);
      System.IO.File.AppendAllText(reportFile, "Total number of crossovers: " +
Chromosome.crossoverCount + Environment.NewLine);
      System.IO.File.AppendAllText(reportFile, "Total number of mutations: " +
Chromosome.mutationCount + Environment.NewLine);
      // don't need this line for report - no constraint violation on my
configuration
      //System.IO.File.AppendAllText(reportFile, "Total number of constraint
violations: " + Chromosome.invalidMutationCount + Environment.NewLine);
    }

    public void GenerationStatisticsFile()
    {
      string statisticsFile = Configuration.StatisticsFilename;
      System.IO.File.AppendAllText(statisticsFile, currentGeneration.ToString() +
"\t" + EAPopulation.MaxFitness + "\t" + EAPopulation.MinFitness + "\t" +
EAPopulation.AvgFitness + Environment.NewLine);
    }
  }
}
```

Α.5 Η κλάση gEP (gEP.cs)

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml;

namespace gEP
{
  /// <summary>
  /// Main class where the EP starts execution.
  /// </summary>
  class gEP
  {
    static void Main(string[] args)
    {
      String configurationFile;
      Random rand;

      if (args.Length > 0)
        configurationFile = args[0];
      else
        configurationFile = "gEP.xml";

      XmlDocument configurationXML = new XmlDocument();
      try
      {
        configurationXML.Load(configurationFile);
      }
      catch (Exception e)
      {
        Console.WriteLine("Program terminated unexpectedly. Please check the
Log.");
        Logger.LogError(e.Message);
        return; // Exits the Program
        //Environment.Exit(1);
      }

      #region Get the configuration and assign it to the Static Class
Configuration

      Logger.LogInfo("Loading Configuration file...");
      /* Loading population size */
      XmlNodeList configurationNodeList =
configurationXML.GetElementsByTagName("PopulationSize");
      if (configurationNodeList.Count > 0)
      {
        try
        {
          Configuration.PopulationSize =
Convert.ToInt32(configurationNodeList[0].InnerText);
          Logger.LogDebug("Loaded " + Configuration.PopulationSize + " as the
population size");
        }
        catch (Exception e)
        {
          Logger.LogWarning("Could not load population size. Using the default
one. (" + e.Message + ")");
        }
      }
```

```
      else
      {
        Logger.LogWarning("No information about population size found, using the
default one (" + Configuration.PopulationSize + ").");
      }

      /* Loading No of runs */
      configurationNodeList = configurationXML.GetElementsByTagName("Runs");
      if (configurationNodeList.Count > 0)
      {
        try
        {
          Configuration.RunCount =
Convert.ToInt32(configurationNodeList[0].InnerText);
          Logger.LogDebug("Loaded " + Configuration.RunCount + " as the Number of
algorithm runs");
        }
        catch (Exception e)
        {
          Logger.LogWarning("Could not load Number of runs. Using the default
one. (" + e.Message + ")");
        }
      }
      else
      {
        Logger.LogWarning("No information about Number of runs found, using the
default one (" + Configuration.RunCount + ").");
      }

      /* Loading brood size */
      configurationNodeList = configurationXML.GetElementsByTagName("BroodSize");
      if (configurationNodeList.Count > 0)
      {
        try
        {
          Configuration.BroodSize =
Convert.ToInt32(configurationNodeList[0].InnerText);
          Logger.LogDebug("Loaded " + Configuration.BroodSize + " as the brood
size");
        }
        catch (Exception e)
        {
          Logger.LogWarning("Could not load brood size. Using the default one. ("
+ e.Message + ")");
        }
      }
      else
      {
        Logger.LogWarning("No information about brood size found, using the
default one (" + Configuration.BroodSize + ").");
      }

      /* Loading maximum number of generations */
      configurationNodeList =
configurationXML.GetElementsByTagName("Generations");
      if (configurationNodeList.Count > 0)
      {
        try
        {
```

```
          Configuration.MaxGenerations =
Convert.ToInt32(configurationNodeList[0].InnerText);
          Logger.LogDebug("Loaded " + Configuration.MaxGenerations + " as the
maximum number of generations.");
        }
        catch (Exception e)
        {
          Logger.LogWarning("Could not load maximum number of generations. Using
the default one. (" + e.Message + ")");
        }
      }
      else
      {
        Logger.LogWarning("No information about maximum number of generetaions
found, using the default one.");
      }

      /* Loading number of genes */
      /* Loading configuration genes (more like a template of genes) */
      configurationNodeList = configurationXML.GetElementsByTagName("Gene");
      if (configurationNodeList.Count > 0)
      {
        Configuration.NumberOfGenes = configurationNodeList.Count;
        Logger.LogDebug("Loaded " + Configuration.NumberOfGenes + " genes for
each chromosome.");
        for (int i = 0; i < configurationNodeList.Count; i++)
        {
          if (configurationNodeList[i].Attributes["bounds"].InnerText == "True")
          {
            try
            {
              int lowerBound =
Convert.ToInt32(configurationNodeList[i].ChildNodes[0].InnerText);
              int upperBound =
Convert.ToInt32(configurationNodeList[i].ChildNodes[1].InnerText);
              Configuration.ConfigGenes.Add(new Gene(lowerBound, upperBound));
              Logger.LogDebug("Loaded gene No " + i.ToString() + " with bounds:
(" + lowerBound.ToString() + ", " + upperBound + ")");
            }
            catch (Exception e)
            {
              Logger.LogWarning("Could not load gene No " + i.ToString() + " due
to a conversion exception (" + e.Message + ") Loading the default configuration
for this gene.");
              Configuration.ConfigGenes.Add(new Gene(null, null));
              Logger.LogDebug("Loaded gene No " + i.ToString() + " with no
bounds");
            }
          }
          else
          {
            Configuration.ConfigGenes.Add(new Gene(null, null));
            Logger.LogDebug("Loaded gene No " + i.ToString() + " with no
bounds");
          }
        }
      }
      else
      {
```

```
        Logger.LogWarning("Could not load any genes. Loading the default genes");
        Configuration.LoadDefaultGenes();
      }

      /* Loading random seed, if any */
      configurationNodeList =
configurationXML.GetElementsByTagName("RandomSeed");
      if ((configurationNodeList.Count > 0) &&
(configurationNodeList[0].Attributes["enabled"].InnerText == "True"))
      {
        try
        {
          int seed = Convert.ToInt32(configurationNodeList[0].InnerText);
          rand = new Random(seed);
          Logger.LogDebug("Loaded configuration for random values with seed: " +
seed.ToString());
        }
        catch (Exception e)
        {
          /* Conversion error. load default configuration */
          Logger.LogWarning("Could not load seed (" +
configurationNodeList[0].InnerText + ") due to a conversion exception (" +
e.Message + ") Loading the default configuration for random.");
          rand = new Random();
          Logger.LogDebug("Loaded configuration for random values with no
seed.");
        }
      }
      else
      {
        rand = new Random();
        Logger.LogDebug("Loaded configuration for random values with no seed.");
      }

      /* Loading Mutation Settings */
      configurationNodeList = configurationXML.GetElementsByTagName("Mutation");
      if ((configurationNodeList.Count > 0) &&
(configurationNodeList[0].Attributes["type"].InnerText.Length > 0) &&
(configurationNodeList[0].Attributes["rate"].InnerText.Length > 0))
      {
        try
        {
          switch (configurationNodeList[0].Attributes["type"].InnerText)
          {
            case "NONE":
              Configuration.MutationType = MutationType.NONE;
              break;
            case "DELTA_MUTATION":
              Configuration.MutationType = MutationType.DELTA_MUTATION;
              break;
            case "GAUSSIAN_MUTATION":
              Configuration.MutationType = MutationType.GAUSSIAN_MUTATION;
              break;
            case "RANDOM_MUTATION":
              Configuration.MutationType = MutationType.RANDOM_MUTATION;
              break;
            default:
              Configuration.MutationType = MutationType.NONE;
              break;
```

```
          }
          // Compute mutation probability from "rate" attribute
          if (configurationNodeList[0].Attributes["rate"].InnerText == "one")
          {
            Configuration.MutationProbability = 1.0F /
Configuration.NumberOfGenes;
            Configuration.MutationDelta =
Convert.ToInt32(configurationNodeList[0].InnerText);
            Logger.LogDebug("Loaded mutation type: " +
Configuration.MutationType.ToString() + " with delta: " +
Configuration.MutationDelta + " and mutation probability: " +
Configuration.MutationProbability);
          }
          else if (configurationNodeList[0].Attributes["rate"].InnerText ==
"all")
          {
            Configuration.MutationProbability = 1.0F;
            Configuration.MutationDelta =
Convert.ToInt32(configurationNodeList[0].InnerText);
            Logger.LogDebug("Loaded mutation type: " +
Configuration.MutationType.ToString() + " with delta: " +
Configuration.MutationDelta + " and mutation probability: " +
Configuration.MutationProbability);
          }
          else if (configurationNodeList[0].Attributes["rate"].InnerText ==
"probability")
          {
            Configuration.MutationProbability =
Convert.ToDouble(configurationNodeList[0].InnerText);
            Logger.LogDebug("Loaded mutation type: " +
Configuration.MutationType.ToString() + " with mutation probability: " +
Configuration.MutationProbability);
          }
          else
            throw new Exception("Mutation rate not properly configured");
        }
        catch (Exception e)
        {
          Configuration.MutationType = MutationType.NONE;
          Configuration.MutationDelta = 0;
          Configuration.MutationProbability = 0.0F;
          Logger.LogWarning("Could not load mutation due to an exception (" +
e.Message + "). Loading the default configuration for mutation.");
          Logger.LogDebug("Loaded mutation type: " +
Configuration.MutationType.ToString() + " with delta: " +
Configuration.MutationDelta + " and mutation probability: " +
Configuration.MutationProbability);
        }
      }
      else
      {
        // Load default configuration for mutation
        Logger.LogDebug("No information about mutation found, using the default
one (none).");
        Configuration.MutationType = MutationType.NONE;
        Configuration.MutationDelta = 0;
        Configuration.MutationProbability = 0;
        Logger.LogDebug("Loaded mutation type: " +
Configuration.MutationType.ToString() + " with delta: " +
```

```
Configuration.MutationDelta + " and mutation probability: " +
Configuration.MutationProbability);
      }

      /* Loading parent selection settings
            * Note that along with parent selection we define the mate selection
type - that will be used in recombination - accordingly*/
      configurationNodeList =
configurationXML.GetElementsByTagName("ParentSelection");
      if (configurationNodeList.Count > 0)
      {
        try
        {
          switch (configurationNodeList[0].InnerText)
          {
            case "FITNESS_PROPORTIONAL":
              Configuration.ParentSelectionType =
SelectionType.FITNESS_PROPORTIONAL;
              Configuration.MateSelectionType =
SelectionType.FITNESS_PROPORTIONAL;
              break;
            case "UNIFORM_STOCHASTIC":
              Configuration.ParentSelectionType =
SelectionType.UNIFORM_STOCHASTIC;
              Configuration.MateSelectionType = SelectionType.UNIFORM_STOCHASTIC;
              break;
            case "UNIFORM_DETERMINISTIC":
              Configuration.ParentSelectionType =
SelectionType.UNIFORM_DETERMINISTIC;
              Configuration.MateSelectionType =
SelectionType.UNIFORM_DETERMINISTIC;
              break;
            case "RANK_PROPORTIONAL":
              Configuration.ParentSelectionType =
SelectionType.RANK_PROPORTIONAL;
              Configuration.MateSelectionType = SelectionType.UNIFORM_STOCHASTIC;
// Using the default one.
              break;
            case "BINARY_TOURNAMENT":
              Configuration.ParentSelectionType =
SelectionType.BINARY_TOURNAMENT;
              Configuration.MateSelectionType = SelectionType.BINARY_TOURNAMENT;
              break;
            case "TRUNCATION":
              Configuration.ParentSelectionType = SelectionType.TRUNCATION;
              Configuration.MateSelectionType = SelectionType.UNIFORM_STOCHASTIC;
// Using the default one.
              break;
            default:
              Configuration.ParentSelectionType =
SelectionType.FITNESS_PROPORTIONAL;
              Configuration.MateSelectionType =
SelectionType.FITNESS_PROPORTIONAL;
              break;
          }
          Logger.LogDebug("Loaded parent selection type: " +
Configuration.ParentSelectionType);
        }
        catch (Exception e)
```

```
        {
          Configuration.ParentSelectionType = SelectionType.FITNESS_PROPORTIONAL;
          Logger.LogWarning("Could not load parent selection type. Using the
default one: " + Configuration.ParentSelectionType + ". (" + e.Message + ")");
        }
      }
      else
      {
        Configuration.ParentSelectionType = SelectionType.FITNESS_PROPORTIONAL;
        Logger.LogWarning("No information about parent selection found, using the
default one (" + Configuration.ParentSelectionType + ").");
      }

      /* Loading survival selection settings */
      configurationNodeList =
configurationXML.GetElementsByTagName("SurvivalSelection");
      if (configurationNodeList.Count > 0)
      {
        try
        {
          switch (configurationNodeList[0].InnerText)
          {
            case "FITNESS_PROPORTIONAL":
              Configuration.SurvivalSelectionType =
SelectionType.FITNESS_PROPORTIONAL;
              break;
            case "UNIFORM_STOCHASTIC":
              Configuration.SurvivalSelectionType =
SelectionType.UNIFORM_STOCHASTIC;
              break;
            case "UNIFORM_DETERMINISTIC":
              Configuration.SurvivalSelectionType =
SelectionType.UNIFORM_DETERMINISTIC;
              break;
            case "RANK_PROPORTIONAL":
              Configuration.SurvivalSelectionType =
SelectionType.RANK_PROPORTIONAL;
              break;
            case "BINARY_TOURNAMENT":
              Configuration.SurvivalSelectionType =
SelectionType.BINARY_TOURNAMENT;
              break;
            case "TRUNCATION":
              Configuration.SurvivalSelectionType = SelectionType.TRUNCATION;
              break;
            default:
              Configuration.SurvivalSelectionType =
SelectionType.FITNESS_PROPORTIONAL;
              break;
          }
          Logger.LogDebug("Loaded survival selection type: " +
Configuration.SurvivalSelectionType);
        }
        catch (Exception e)
        {
          Configuration.SurvivalSelectionType = SelectionType.UNIFORM_STOCHASTIC;
          Logger.LogWarning("Could not load survival selection type. Using the
default one: " + Configuration.SurvivalSelectionType + ". (" + e.Message + ")");
        }
```

```
        }
        else
        {
          Configuration.SurvivalSelectionType = SelectionType.UNIFORM_STOCHASTIC;
          Logger.LogWarning("No information about survival selection found, using
the default one (" + Configuration.SurvivalSelectionType + ").");
        }

        /* Loading Crossover Settings */
        configurationNodeList = configurationXML.GetElementsByTagName("Crossover");
        if (configurationNodeList.Count > 0)
        {
          try
          {
            switch (configurationNodeList[0].Attributes["type"].InnerText)
            {
              case "ONE_POINT":
                Configuration.CrossoverType = CrossoverType.ONE_POINT;
                break;
              case "TWO_POINT":
                Configuration.CrossoverType = CrossoverType.TWO_POINT;
                break;
              case "UNIFORM":
                Configuration.CrossoverType = CrossoverType.UNIFORM;
                break;
            }
            Configuration.CrossoverProbability =
Convert.ToDouble(configurationNodeList[0].InnerText);
            Logger.LogDebug("Loaded crossover type: " + Configuration.CrossoverType
+ " with crossover probability: " + Configuration.CrossoverProbability);
          }
          catch (Exception e)
          {
            Configuration.CrossoverType = CrossoverType.ONE_POINT;
            Logger.LogWarning("Could not load crossover type. Using the default
one: " + Configuration.CrossoverType + ". (" + e.Message + ")");
          }
        }
        else
        {
          Logger.LogWarning("No information about crossover type found, using the
default one (" + Configuration.CrossoverType + ").");
        }

        /* Loading Connection Settings */
        configurationNodeList =
configurationXML.GetElementsByTagName("ConnectionString");
        if (configurationNodeList.Count > 0)
        {
          try
          {
            Configuration.ConnectionString = configurationNodeList[0].InnerText;
            Logger.LogDebug("Loaded connection string: " +
Configuration.ConnectionString);
          }
          catch (Exception e)
          {
            Configuration.ConnectionString = "Data Source=localhost;Initial
Catalog=KDD99;Integrated Security=True";
```

```
            Logger.LogWarning("Could not load connection string. Using the default
one: " + Configuration.ConnectionString + ". (" + e.Message + ")");
          }
        }
        else
        {
          Configuration.ConnectionString = "Data Source=localhost;Initial
Catalog=KDD99;Integrated Security=True";
          Logger.LogWarning("No information about connection found, using the
default one (" + Configuration.ConnectionString + ").");
        }

        Logger.LogInfo("Configuration file loaded.");

        #endregion


        /* Starting execution of EP from here */
        Logger.LogInfo("Starting Algorithm Execution");
        // Prepare connection to DB Server
        DAL.OpenConnection();
        // create report file name
        string timestamp = DateTime.Now.ToString("yyyyMMdd_HHmmss");
        Configuration.ReportFilename = timestamp + "_" +
Configuration.ReportFilename + ".txt";
        Configuration.StatisticsFilename = timestamp + "_" +
Configuration.StatisticsFilename + ".txt";
        Configuration.PoolFilename = timestamp + "_" + Configuration.PoolFilename +
".txt";
        // Create configuration file in order to log the parameters of the genetic
algorithm
        string confFileName = timestamp + "_" + "conf.txt";
        System.IO.File.AppendAllText(confFileName, "Runs: " +
Configuration.RunCount.ToString() + Environment.NewLine);
        System.IO.File.AppendAllText(confFileName, "PopulationSize: " +
Configuration.PopulationSize.ToString() + Environment.NewLine);
        System.IO.File.AppendAllText(confFileName, "BroodSize: " +
Configuration.BroodSize.ToString() + Environment.NewLine);
        System.IO.File.AppendAllText(confFileName, "Generations: " +
Configuration.MaxGenerations.ToString() + Environment.NewLine);
        System.IO.File.AppendAllText(confFileName, "Crossover: " +
Configuration.CrossoverType.ToString() + " probability: " +
Configuration.CrossoverProbability + Environment.NewLine);
        System.IO.File.AppendAllText(confFileName, "Mutation: " +
Configuration.MutationType.ToString() + " probability: " +
Configuration.MutationProbability + Environment.NewLine);
        System.IO.File.AppendAllText(confFileName, "ParentSelection: " +
Configuration.ParentSelectionType.ToString() + Environment.NewLine);
        System.IO.File.AppendAllText(confFileName, "SurvivalSelection: " +
Configuration.SurvivalSelectionType.ToString());
        // Initialize and run Evolutionary algorithm
        Configuration.LoadPossibleValues();
        for (int i = 1; i <= Configuration.RunCount; i++)
        {
          if (Configuration.RunCount > 1)
          {
            Logger.LogInfo("---------- Executing Run " + i + " ----------");
            Console.WriteLine();
            Console.WriteLine("---------- Executing Run " + i + " ----------");
```

```
            }
            EA applyEA = new EA();
            applyEA.Run(rand);
        }
    }
  }
}
```

A.6 Η κλάση Configuration (Configuration.cs)

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text;
using System.Data;

namespace gEP
{
  /// <summary>
  /// A static class that represents the configuration of the running EP.
  /// </summary>
  static class Configuration
  {
    private static int runCount = 1;
    private static int populationSize = 10;
    private static int maxGenerations = 10;
    private static int numberOfGenes = 1;
    private static List<Gene> configGenes = new List<Gene>();
    private static MutationType mutationType;
    private static int mutationDelta;
    private static double mutationProbability;
    private static SelectionType parentSelectionType;
    private static SelectionType mateSelectionType;
    private static SelectionType survivalSelectionType;
    private static int broodSize = 1;
    private static CrossoverType crossoverType;
    private static double crossoverProbability;
    private static string connectionString;
    private static ArrayList duration;
    private static ArrayList src_bytes;
    private static ArrayList dst_host_srv_serror_rate;
    private static ArrayList distinctDuration;
    //private static ArrayList distinctSrc_bytes;
    //private static ArrayList distinctDst_host_srv_serror_rate;
    private static string reportFilename = "Report"; //without extension!
    private static string statisticsFilename = "Statistics"; //without extension
    private static string poolFilename = "pool"; //without extension

    private static bool haveNextNextGaussian = false;
    private static double nextNextGaussian;

    #region Public Properties

    public static int RunCount
    {
      get { return runCount; }
      set { runCount = value; }
    }

    public static int PopulationSize
    {
      get { return populationSize; }
      set { populationSize = value; }
    }

    public static int MaxGenerations
    {
      get { return maxGenerations; }
```

```
    set { maxGenerations = value; }
  }

  public static int NumberOfGenes
  {
    get { return numberOfGenes; }
    set { numberOfGenes = value; }
  }

  public static List<Gene> ConfigGenes
  {
    get { return configGenes; }
    set { configGenes = value; }
  }

  public static MutationType MutationType
  {
    get { return mutationType; }
    set { mutationType = value; }
  }

  public static int MutationDelta
  {
    get { return mutationDelta; }
    set { mutationDelta = value; }
  }

  public static double MutationProbability
  {
    get { return mutationProbability; }
    set { mutationProbability = value; }
  }

  public static SelectionType ParentSelectionType
  {
    get { return parentSelectionType; }
    set { parentSelectionType = value; }
  }

  public static SelectionType MateSelectionType
  {
    get { return mateSelectionType; }
    set { mateSelectionType = value; }
  }

  public static SelectionType SurvivalSelectionType
  {
    get { return survivalSelectionType; }
    set { survivalSelectionType = value; }
  }

  public static int BroodSize
  {
    get { return broodSize; }
    set { broodSize = value; }
  }

  public static CrossoverType CrossoverType
  {
```

```
        get { return crossoverType; }
        set { crossoverType = value; }
    }

    public static double CrossoverProbability
    {
        get { return crossoverProbability; }
        set { crossoverProbability = value; }
    }

    public static string ConnectionString
    {
        get { return connectionString; }
        set { connectionString = value; }
    }

    public static ArrayList Duration
    {
        get { return duration; }
        //set { duration = value; }
    }

    public static ArrayList Src_bytes
    {
        get { return src_bytes; }
        //set { src_bytes = value; }
    }

    public static ArrayList Dst_host_srv_serror_rate
    {
        get { return dst_host_srv_serror_rate; }
        //set { dst_host_srv_serror_rate = value; }
    }

    /*public static ArrayList DistinctDuration
    {
        get { return distinctDuration; }
        //set { duration = value; }
    }

    public static ArrayList DistinctSrc_bytes
    {
        get { return distinctSrc_bytes; }
        //set { src_bytes = value; }
    }

    public static ArrayList DistinctDst_host_srv_serror_rate
    {
        get { return distinctDst_host_srv_serror_rate; }
        //set { dst_host_srv_serror_rate = value; }
    }*/

    public static string ReportFilename
    {
        get { return reportFilename; }
        set { reportFilename = value; }
    }

    public static string StatisticsFilename
```

```
    {
      get { return statisticsFilename; }
      set { statisticsFilename = value; }
    }

    public static string PoolFilename
    {
      get { return poolFilename; }
      set { poolFilename = value; }
    }

    #endregion

    /// <summary>
    /// This is the default action when no parameters
    /// are specified in the configuration XML file
    /// </summary>
    public static void LoadDefaultGenes()
    {
      /* Create genes with no bounds using the field NumberOfGenes
       * for the number of the genes that will be created */
      for (int i = 0; i < NumberOfGenes; i++)
      {
        ConfigGenes.Add(new Gene(null, null));
      }
    }

    /// <summary>
    /// Loads the possible values that each gene can have from database
    /// </summary>
    public static void LoadPossibleValues()
    {
      int rowCount = 0;
      System.Data.SqlClient.SqlDataReader dataReader;
      //System.Data.SqlClient.SqlDataReader dataReader1;

      // Load duration ArrayList with possible values
      //rowCount = (int)DAL.ExecuteScalar("select count(distinct duration) from
training_set");
      rowCount = 1000;
      dataReader = DAL.ExecuteQuery("select duration from training_set order by
1");
      duration = new ArrayList(rowCount);
      distinctDuration = new ArrayList();
      // Loop through data reader to add items to the ArrayList
      while (dataReader.Read())
      {
        duration.Add(int.Parse(dataReader[0].ToString()));
      }
      dataReader.Close();

      /*dataReader1 = DAL.ExecuteQuery("select distinct duration from
training_set order by 1");
      distinctDuration = new ArrayList();
      // Loop through data reader to add items to the ArrayList
      while (dataReader1.Read())
      {
          distinctDuration.Add(int.Parse(dataReader1[0].ToString()));
      }
```

```
      dataReader1.Close();*/

      // Load src_bytes ArrayList with possible values
      //rowCount = (int)DAL.ExecuteScalar("select count(distinct src_bytes) from
training_set");
      dataReader = DAL.ExecuteQuery("select src_bytes from training_set order by
1");
      src_bytes = new ArrayList(rowCount);
      // Loop through data reader to add items to the ArrayList
      while (dataReader.Read())
      {
        src_bytes.Add(int.Parse(dataReader[0].ToString()));
      }
      dataReader.Close();

      /*dataReader1 = DAL.ExecuteQuery("select distinct src_bytes from
training_set order by 1");
      distinctSrc_bytes = new ArrayList();
      // Loop through data reader to add items to the ArrayList
      while (dataReader1.Read())
      {
          distinctSrc_bytes.Add(int.Parse(dataReader1[0].ToString()));
      }
      dataReader1.Close();*/

      // Load dst_host_srv_serror_rate ArrayList with possible values
      //rowCount = (int)DAL.ExecuteScalar("select count(distinct
dst_host_srv_serror_rate) from training_set");
      dataReader = DAL.ExecuteQuery("select dst_host_srv_serror_rate from
training_set order by 1");
      dst_host_srv_serror_rate = new ArrayList(rowCount);
      // Loop through data reader to add items to the ArrayList
      while (dataReader.Read())
      {
        dst_host_srv_serror_rate.Add(int.Parse(dataReader[0].ToString()));
      }
      dataReader.Close();

      /*dataReader1 = DAL.ExecuteQuery("select distinct dst_host_srv_serror_rate
from training_set order by 1");
      distinctDst_host_srv_serror_rate = new ArrayList();
      // Loop through data reader to add items to the ArrayList
      while (dataReader1.Read())
      {

distinctDst_host_srv_serror_rate.Add(int.Parse(dataReader1[0].ToString()));
      }
      dataReader1.Close();*/
    }

    /// <summary>
    /// Returns the next pseudorandom, Gaussian ("normally") distributed double
value with mean 0.0 and standard deviation 1.0 from this random number
generator's sequence.
    /// </summary>
    /// <param name="rand">The random number generator</param>
    /// <returns>The next pseudorandom, Gaussian ("normally") distributed double
value with mean 0.0 and standard deviation 1.0</returns>
    public static double NextGaussian(Random rand)
```

```
    {
      if (haveNextNextGaussian)
      {
        haveNextNextGaussian = false;
        return nextNextGaussian;
      }
      else
      {
        double v1, v2, s;
        do
        {
          v1 = 2 * rand.NextDouble() - 1; // between -1.0 and 1.0
          v2 = 2 * rand.NextDouble() - 1; // between -1.0 and 1.0
          s = v1 * v1 + v2 * v2;
        } while (s >= 1 || s == 0);
        double multiplier = Math.Sqrt(-2 * Math.Log(s) / s);
        nextNextGaussian = v2 * multiplier;
        haveNextNextGaussian = true;
        return v1 * multiplier;
      }
    }

  }

  /// <summary>
  /// Enumerator that stores mutation types.
  /// </summary>
  public enum MutationType
  {
    /// <summary>
    /// No mutation
    /// </summary>
    NONE,
    /// <summary>
    /// Delta mutation
    /// </summary>
    DELTA_MUTATION,
    /// <summary>
    /// Random mutation
    /// </summary>
    RANDOM_MUTATION,
    /// <summary>
    /// Gaussian mutation
    /// </summary>
    GAUSSIAN_MUTATION
  }

  /// <summary>
  /// Enumerator that stores parent selection types
  /// </summary>
  public enum SelectionType
  {
    /// <summary>
    /// Fitness Proportional selection
    /// </summary>
    FITNESS_PROPORTIONAL,
    /// <summary>
    /// Uniform Stochastic selection
    /// </summary>
```

```
    UNIFORM_STOCHASTIC,
    /// <summary>
    /// Uniform Deterministic selection
    /// </summary>
    UNIFORM_DETERMINISTIC,
    /// <summary>
    /// Rank Proportional selection
    /// </summary>
    RANK_PROPORTIONAL,
    /// <summary>
    /// Binary Tournament selection
    /// </summary>
    BINARY_TOURNAMENT,
    /// <summary>
    /// Truncation selection
    /// </summary>
    TRUNCATION
  }

  /// <summary>
  /// Enumerator that stores crossover types.
  /// </summary>
  public enum CrossoverType
  {
    /// <summary>
    /// One point crossover
    /// </summary>
    ONE_POINT,
    /// <summary>
    /// Two point crossover
    /// </summary>
    TWO_POINT,
    /// <summary>
    /// Uniform crossover
    /// </summary>
    UNIFORM
  }
}
```

Α.7 Η κλάση DAL (DAL.cs)

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Data;
using System.Data.SqlClient;

namespace gEP
{
  static class DAL
  {
    private static string mSQLConnectionString = "";
    private static IDbConnection mSQLConnection;
    private static IDbCommand mSQLCommand;


    public static void InitConnectionString()
    {
      mSQLConnectionString = Configuration.ConnectionString;
    }

    public static SqlConnection CreateNewConnection()
    {
      InitConnectionString();
      System.Data.IDbConnection newSQLConnection;
      try
      {
        newSQLConnection = new SqlConnection(mSQLConnectionString);
      }
      catch (Exception e)
      {
        Logger.LogError("An error occured while trying to create a database
connection. " + e.Message);
        return null;
      }
      return (SqlConnection)newSQLConnection;
    }

    public static SqlConnection OpenConnection()
    {
      try
      {
        mSQLConnection = CreateNewConnection();
        mSQLConnection.Open();
        mSQLCommand = mSQLConnection.CreateCommand();
      }
      catch (Exception e)
      {
        Logger.LogError("An error occured while trying to open a database
connection. " + e.Message);
        return null;
      }
      return (SqlConnection)mSQLConnection;
    }

    public static int CloseConnection()
    {
      if (mSQLConnection == null)
        return 0;
```

```
      try
      {
        mSQLConnection.Close();
      }
      catch (Exception e)
      {
        Logger.LogWarning("An error occured while trying to close a database
connection. " + e.Message);
        return 0;
      }
      return 1;
    }

    public static SqlDataReader ExecuteQuery(string selectString)
    {
      try
      {
        mSQLCommand.CommandText = selectString;
        mSQLCommand.CommandType = CommandType.Text;
        SqlDataReader reader = (SqlDataReader)mSQLCommand.ExecuteReader();
        return reader;
      }
      catch (Exception e)
      {
        Logger.LogError("An error occured while trying to execute a database
query. " + e.Message);
        return null;
      }
    }

    public static object ExecuteScalar(string selectString)
    {
      object obj = null;
      try
      {
        mSQLCommand.CommandText = selectString;
        mSQLCommand.CommandType = CommandType.Text;
        obj = mSQLCommand.ExecuteScalar();
      }
      catch (Exception e)
      {
        Logger.LogError("An error occured while trying to execute a scalar
database query. " + e.Message);
      }
      return obj;
    }
  }
}
```

A.8 Η κλάση Logger (Logger.cs)

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace gEP
{
  /// <summary>
  /// This static class is a utility class for logging runs of the program
  /// </summary>
  static class Logger
  {
    /// <summary>
    /// This property keeps the name of the log file
    /// </summary>
    private static string LogFile = "gEPLog.txt";

    /// <summary>
    /// Log a Debug level information
    /// </summary>
    /// <param name="text">text to write to log file</param>
    public static void LogDebug(string text)
    {
      text = DateTime.Now.ToString("yyyy/MM/dd HH:mm:ss") + " [DEBUG] " + text +
Environment.NewLine;
      File.AppendAllText(LogFile, text);
    }

    /// <summary>
    /// Log an Info level information
    /// </summary>
    /// <param name="text">text to write to log file</param>
    public static void LogInfo(string text)
    {
      text = DateTime.Now.ToString("yyyy/MM/dd HH:mm:ss") + " [INFO] " + text +
Environment.NewLine;
      File.AppendAllText(LogFile, text);
    }

    /// <summary>
    /// Log a Warning level information
    /// </summary>
    /// <param name="text">text to write to log file</param>
    public static void LogWarning(string text)
    {
      text = DateTime.Now.ToString("yyyy/MM/dd HH:mm:ss") + " [WARNING] " + text
+ Environment.NewLine;
      File.AppendAllText(LogFile, text);
    }

    /// <summary>
    /// Log an Error level information
    /// </summary>
    /// <param name="text">text to write to log file</param>
    public static void LogError(string text)
    {
      text = DateTime.Now.ToString("yyyy/MM/dd HH:mm:ss") + " [ERROR] " + text +
Environment.NewLine;
```

```
      File.AppendAllText(LogFile, text);
    }
  }
}
```