ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ
ΤΕΙ ΗΠΕΙΡΟΥ

ΑΡΤΑ, ΣΕΠΤΕΜΒΡΙΟΣ 2017

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΓΕΝΕΤΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ ΚΑΙ ΓΡΑΜΜΑΤΙΚΗ ΕΞΕΛΙΞΗ - ΜΕΛΕΤΗ ΚΑΙ ΕΦΑΡΜΟΓΗ ΣΕ ΠΑΙΧΝΙΔΙΑ ΓΡΙΦΩΝ

ΝΙΚΟΛΑΟΣ ΚΟΡΝΕΛΑΚΗΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΣΤΑΥΡΟΣ Π. ΑΔΑΜ

ΤΕΙ ΗΠΕΙΡΟΥ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΓΕΝΕΤΙΚΟΣ ΑΛΓΟΡΙΘΜΟΣ ΚΑΙ ΓΡΑΜΜΑΤΙΚΗ ΕΞΕΛΙΞΗ – ΜΕΛΕΤΗ ΚΑΙ ΕΦΑΡΜΟΓΗ ΣΕ ΠΑΙΧΝΙΔΙΑ ΓΡΙΦΩΝ

ΝΙΚΟΛΑΟΣ ΚΟΡΝΕΛΑΚΗΣ

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ
ΣΤΑΥΡΟΣ Π. ΑΔΑΜ

ΑΡΤΑ, ΣΕΠΤΕΜΒΡΙΟΣ 2017

GENETIC ALGORITHM AND GRAMMATICAL EVOLUTION – STUDY AND APPLICATION IN PUZZLE GAMES

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Κορνελάκης, Νικόλαος

Υπογραφή

Ευχαριστίες

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή μου, κ. Σταύρο Π. Αδάμ, για την εμπιστοσύνη που μου έδειξε και την υπομονή του, καθώς και για την υποστήριξη και γενικότερα την πολύτιμη βοήθειά του σε αυτή την πτυχιακή εργασία.

Περίληψη

Τα παιχνίδια γρίφων αποτελούν μια διακεκριμένη κατηγορία προβλημάτων συνδυαστικής βελτιστοποιήσης, τα οποία συχνά είναι δύσκολο να λυθούν λόγω του μεγέθους του χώρου αναζήτησης, ο οποίος αυξάνεται εκθετικά με τον αριθμό των παραμέτρων του προβληματος. Στην βιβλιογραφία έχουν προταθεί διάφορες προσεγγίσεις και αλγόριθμοι για την επίλυση τέτοιων προβλημάτων. Μεταξύ αυτών των μεθόδων, οι ευρετικές προσεγγίσεις και οι σχετικοί αλγόριθμοι, όπως ο Γενετικός Αλγόριθμος, ο αλγόριθμος Βελτιστοποίησης Σμήνους Σωματιδίων και άλλοι, φαίνεται να είναι μια ενδιαφέρουσα κατηγορία μεθόδων. Αυτή η εργασία παρουσιάζει μια μελέτη χρήσης της Γραμματικής Εξέλιξης για την επίλυση τέτοιων προβλημάτων.

Για να μελετήσουμε τη συμπεριφορά της Γραμματικής Εξέλιξης σε τέτοια προβλήματα, πραγματοποιήσαμε μια σειρά πειραμάτων σε δύο πολύ γνωστά παιχνίδια γρίφων, το Mastermind και το SameGame. Αυτά τα παιχνίδια επιλέχθηκαν καθώς έχουν έναν ισχυρό συνδυαστικό χαρακτήρα ενώ ταυτόχρονα το μέγεθος του χώρου αναζήτησης καθιστά εύκολη την παρακολούθηση της λειτουργίας της προτεινόμενης διαδικασίας βελτιστοποίησης. Η εφαρμογή της Γραμματικής Εξέλιξης στα παραπάνω προβλήματα έγινε χρησιμοποιώντας το περιβάλλον GEVA. Για να κατανοήσουμε καλύτερα και να γνωρίσουμε το περιβάλλον, χρησιμοποιήσαμε αρχικά το πρόβλημα των Πύργων του Ανόι.

Μια πρόχειρη εκτίμηση των αποτελεσμάτων και των στατιστικών που υπολογίστηκαν έδειξε μια αρκετά καλή συμπεριφορά της Γραμματικής Εξέλιξης για την επίλυση των επιλεγμένων προβλημάτων. Μπορούμε λοιπόν να συμπεράνουμε ότι η Γραμματική Εξέλιξη είναι κατάλληλη για την επίλυση του συγκεκριμένου είδους προβλημάτων, ενώ φαίνεται πολλά υποσχόμενη η εφαρμογή της μεθόδου αυτής σε παιχνίδια γρίφων υψηλότερων διαστάσεων και σε άλλα προβλήματα, όπως παιχνίδια συνδυαστικής βελτιστοποίησης ή, γενικά, προβλήματα συνδυαστικής βελτιστοποίησης.

Λέξεις-κλειδιά: συνδυαστική βελτιστοποίηση, Γενετικός Αλγόριθμος, Γενετικός Προγραμματισμός, Γραμματική Εξέλιξη, γραμματική, αντικειμενική συνάρτηση.

Abstract

Puzzle games constitute a distinguished class of combinatorial optimization problems which, often, are rather difficult to solve because of the size of the search space which increases exponentially with the number of the problem parameters. Several different approaches and algorithms have been proposed in the literature for solving such problems. Among these methods, heuristic approaches and related algorithms, such as the Genetic Algorithm, Particle Swarm Optimization algorithm, and others seem to constitute an interesting class of methods. This thesis presents a study of the use of Grammatical Evolution to solve such problems.

In order to study the behavior of Grammatical Evolution in such problems, we carried out a number of experiments on two well known puzzle games, namely, Mastermind and SameGame. These games were selected as they have a strong combinatorial character while at the same time the size of the search space makes it easy to track the operation of the proposed optimization procedure. The application of Grammatical Evolution to the above problems was done using the GEVA environment. In order to better understand and be familiar with the environment, we initially used the problem of the Hanoi Towers.

A rough estimation of the results obtained and the statistics calculated showed a fairly good behavior of Grammar Evolution for solving the chosen problems. So, we may conclude that Grammatical Evolution is well suited for solving the specific kind of problems while it seems promising to apply the method to higher dimension puzzle games and other problems such as combinatorial optimization games or, generally, combinatorial optimization problems.

Keywords: combinatorial optimization, Genetic Algorithm, Genetic Programming, Grammatical Evolution, grammar, fitness function.

Περιεχόμενα

Λίστα Πινάκων

Λίστα Εικόνων

Πίνακας Συντομογραφιών

ΒΣΣ……..……...Βελτιστοποίηση Σμήνους Σωματιδίων

ΓΑ…….Γενετικός Αλγόριθμος

ΓΠ……...…...Γενετικός Προγραμματισμός

ΓΕ……...…….Γραμματική Εξέλιξη

ΥΝ……..…...…Υπολογιστική Νοημοσύνη

ΕΥ……...……Εξελικτικός Υπολογισμός

1.Εισαγωγή

Τα προβλήματα συνδυαστικής βελτιστοποίησης είναι συνήθως δύσκολο να λυθούν με μία μεθοδολογία σε λογικό χρόνο, λόγω του μεγάλου χώρου αναζήτησης των λύσεων που αυξάνει την πολυπλοκότητά τους. Στην πλειοψηφία τους οι μέθοδοι που χρησιμοποιούνται για την επίλυση τέτοιων προβλημάτων είναι ευρετικές.

Στην βιβλιογραφία μπορεί κανείς να βρει αρκετές μεθόδους για την επίλυση προβλημάτων συνδυαστικής βελτιστοποίησης. Στην εργασία [1] γίνεται μία μελέτη για την επίλυση τέτοιων προβλημάτων και πιο συγκεκριμένα του προβλήματος εξισορρόπησης της γραμμής συναρμολόγησης (Assembly Line Balancing Problem) με χρήση του ΓΑ. Άλλη μία μέθοδος που προτάθηκε στην εργασία [2] είναι μία υπερ-ευρετική προσέγγιση ΓΠ. Επίσης έχουν προταθεί αρκετές παραλλαγές της μεθόδου ΒΣΣ (Particle Swarm Optimization) για την επίλυση προβλημάτων συνδυαστικής βελτιστοποίησης. Μερικές από αυτές είναι η χαώδης ΒΣΣ που φαίνεται στο άρθρο [3], η συνδυαστική ΒΣΣ που εφαρμόζεται στα προβλήματα διαιρετικής ομαδοποίησης (partitional clustering) και δρομολόγησης κλειδώματος ροών (blocking flowshop scheduling) στα [4] και [5] αντίστοιχα. Αρκετές διαφορετικές προσεγγίσεις διαφορο-εξελικτικών αλγόριθμων έχουν επίσης προταθεί για την επίλυση τέτοιων προβλημάτων. Μία τέτοια προσέγγιση διατυπώνεται στο άρθρο [6]. Τέλος η προσέγγιση της αποικίας μυρμηγκιών (ant colony) ή της τεχνητής αποικίας μελισσών (artificial bee colony) είναι ακόμα δύο μέθοδοι που έχουν μελετηθεί και εφαρμοστεί επάνω σε προβλήματα συνδυαστικής βελτιστοποίησης. Μερικά ενδεικτικά άρθρα για αυτές είναι τα [7], [8], [9], [10] και [11].

Η ΓΕ είναι μία ευέλικτη τεχνική λόγω της σπονδυλωτής μορφής της. Έτσι δίνει την δυνατότητα χρήσης διάφορων στρατηγικών αναζήτησης είτε εξελικτικές είτε άλλες ευρετικές (στοχαστικές ή ντετερμινιστικές). Επίσης με τη χρήση της ΓΕ μπορεί κάποιος να παράξει λύσεις σε οποιαδήποτε γλώσσα είτε προγραμματισμού είτε φυσική ή ακόμα και σε ένα υποσύνολο μίας γλώσσας. Τέλος, η διαδικασία χαρτογράφησης, η οποία αναλύεται στο υποκεφάλαιο 4.1, επιτρέπει στους τελεστές αναζήτησης να εκτελούνται στον γονότυπο, σε μερικώς παραγόμενους φαινοτύπους και ακόμα και στα πλήρως σχηματισμένα φαινοτυπικά δέντρα παραγωγής τα ίδια σε αντίθεση με τον ΓΠ όπου εκτελούνται μόνο σε δέντρα. Όλα τα παραπάνω καθιστούν την ΓΕ μία πολύ καλή και

ευέλικτη μέθοδο επίλυσης προβλημάτων που είναι και ο σημαντικότερος λόγος να την χρησιμοποιήσει κάποιος.

Στο επόμενο κεφάλαιο γίνεται μία συνοπτική περιγραφή κάποιων βασικών εννοιών που αποτελούν το θεωρητικό πλαίσιο της πτυχιακής εργασίας. Στο τρίτο κεφάλαιο γίνεται αναφορά σε δύο τεχνικές που βοήθησαν στην διαμόρφωση της ΓΕ, τον ΓΑ και τον ΓΠ ενώ στο τέταρτο κεφάλαιο αναλύεται σε βάθος ο τρόπος λειτουργίας της ΓΕ. Στο πέμπτο κεφάλαιο γίνεται μία περιγραφή του περιβάλλοντος που χρησιμοποιήσαμε, των προβλημάτων που επιλέξαμε για να λύσουμε με την ΓΕ, αλλά και της διαδικασία που ακολουθήσαμε. Τέλος στο έκτο κεφάλαιο αναφέρονται κάποια γενικά συμπεράσματα στα οποία καταλήξαμε μετά από την ολοκλήρωση της παρούσας εργασίας.

2. Θεωρητικό πλαίσιο

2.1 Συνδυαστική Βελτιστοποίηση

Η συνδυαστική βελτιστοποίηση αποτελεί ένα κλάδο της βελτιστοποίησης στα εφαρμοσμένα μαθηματικά και στην επιστήμη των υπολογιστών, ο οποίος συνενώνει πολλά άλλα επιστημονικά πεδία, όπως της επιχειρησιακής έρευνας, της αλγοριθμικής θεωρίας, της υπολογιστικής πολυπλοκότητας, αλλά και της τεχνητής νοημοσύνης. Αλγόριθμοι συνδυαστικής βελτιστοποίησης εφαρμόζονται ώστε να λυθούν προβλήματα που θεωρούνται αρκετά δύσκολα, λόγω του μεγάλου χώρου αναζήτησης. Τέτοια προβλήματα είναι και τα NP-hard στα οποία οι αλγόριθμοι συνδυαστικής βελτιστοποίησης βρίσκουν πολλές εφαρμογές.

Ο στόχος λοιπόν σε ένα πρόβλημα βελτιστοποίησης είναι η μεγιστοποίηση ή η ελαχιστοποίηση μίας συνάρτησης, η οποία συναντάται στη βιβλιογραφία με διαφορετικές ονομασίες όπως αντικειμενική συνάρτηση, συνάρτηση ικανότητας ή συνάρτηση κόστους.

Υπάρχουν πολλά καθημερινά προβλήματα που εντάσσονται σε αυτή την κατηγορία προβλημάτων, όπως για παράδειγμα την εύρεση της συντομότερης διαδρομής σε μια μετακίνηση, ή την καλύτερη κατανομή εργασιών ώστε να ολοκληρωθούν το συντομότερο δυνατόν. Αυτό δείχνει και την σημαντικότητα αυτού του επιστημονικού πεδίου. Έτσι η επίλυση των προβλημάτων συνδυαστικής βελτιστοποίησης επιτυγχάνεται με τη μείωση του χώρου αναζήτησης, καθώς και με τη χρήση αποτελεσματικότερων τρόπων αναζήτησης.

Αναλυτικότερες πληροφορίες για την συνδυαστική βελτιστοποίηση, υπάρχουν στο [12].

2.2 Υπολογιστική Νοημοσύνη

Η ΥΝ (Computational Intelligence) εμφανίστηκε περίπου το 1990 ως τίτλος μίας ομάδας μεθοδολογιών. Συγκεκριμένα, ως ΥΝ ορίζουμε μια συνεχώς εξελισσόμενη συνέργεια μεθοδολογιών επεξεργασίας αριθμητικών δεδομένων, οι οποίες είναι υλοποιήσιμες σε υπολογιστή για τη λήψη αποφάσεων κοινής λογικής. Υπό αυτήν την έννοια, μια τεχνολογία ΥΝ αναμένεται να είναι συμβατή με αρχές της κοινής λογικής.

Σύμφωνα με το IEEE Computational Intelligence Society η ΥΝ ορίζεται ως «Η θεωρία, ο σχεδιασμός, η εφαρμογή και η ανάπτυξη υποδειγματικών υπολογιστικών βιολογικών και γλωσσικών παραδειγμάτων με έμφαση σε νευρωνικά δίκτυα, συστήματα σύνδεσης, γενετικούς αλγόριθμους, εξελικτικό προγραμματισμό, ασαφή συστήματα και υβριδικά ευφυή συστήματα στα οποία περιέχονται αυτά τα παραδείγματα.» [13].

Είναι ενδιαφέρον να αναφερθεί η σχέση της ΥΝ με την τεχνητή νοημοσύνη (Artificial Intelligence). Η ΥΝ βασίζεται στην επεξεργασία αριθμών. Συγκεκριμένα, μια μεθοδολογία ΥΝ μπορεί να λαμβάνει αριθμητικά δεδομένα από ένα ή περισσότερα ηλεκτρονικά όργανα μέτρησης, ενώ όταν θεωρούνται μη-αριθμητικά δεδομένα, η επεξεργασία τους στα πλαίσια της ΥΝ τελικά ανάγεται στην επεξεργασία αριθμών.

Αντιθέτως, η τεχνητή νοημοσύνη βασίζεται στην επεξεργασία συμβόλων. Μια κλασική μεθοδολογία ΥΝ τυπικά προσομοιώνει παραμετρικά κάποια λειτουργία ενός βιολογικού οργάνου/οργανισμού. Για παράδειγμα, τα τεχνητά νευρωνικά δίκτυα (Artificial Neural Network), μια κλασική μεθοδολογία ΥΝ, προσομοιώνουν τη λειτουργία του εγκεφάλου. Μια άλλη τέτοια μεθοδολογία, τα ασαφή συστήματα (Fuzzy Systems), προσομοιώνει τη λειτουργία της ομιλούμενης γλώσσας. Ο ΕΥ (Evolutionary Computation), μια ακόμα μεθοδολογία ΥΝ, προσομοιώνει τη διαδικασία της φυσικής (Δαρβινικής) επιλογής. Γίνεται αναλυτικότερη περιγραφή του στο επόμενο υποκεφάλαιο. Οι προαναφερθείσες τρεις μεθοδολογίες αποτελούν το περιεχόμενο της κλασικής ΥΝ. Από το συνδυασμό μεθοδολογιών της, ή/και την επιλεκτική αντικατάστασή τους, προέκυψαν εναλλακτικές μεθοδολογίες. Παραδείγματα τέτοιων μεθοδολογιών αποτελούν τα νευρο-ασαφή συστήματα, τα δίκτυα ακτινωτής βάσης, οι μηχανές διανυσμάτων στήριξης, οι γνωσιακοί χάρτες κ.λπ.

Οι εφαρμογές ΥΝ γενικά εμπίπτουν σε μια από τις ακόλουθες τρεις κατηγορίες: (α) ομαδοποίηση (clustering), (β) κατηγοριοποίηση (classification), (γ) παλινδρόμηση (στατιστική) (regression (statistical)).

Οι παραπάνω πληροφορίες αναφέρονται εκτενέστερα στο βιβλίο [14].

2.3 Εξελικτικός Υπολογισμός

2.3.1 Τρόπος λειτουργίας

Οι αλγόριθμοι ΕΥ είναι ένα υποσύνολο ενός συνόλου αλγόριθμων βελτιστοποίησης που ονομάζονται εμπνευσμένοι από την φύση (nature inspired).

Η έννοια του ΕΥ πρωτοεμφανίστηκε τις δεκαετίες του 1950 και 1960. Ο όρος αυτός χρησιμοποιήθηκε για να περιγράψει ένα σύνολο αλγόριθμων, όπου κάθε αλγόριθμος βασίζεται σε έναν πληθυσμό υποψήφιων λύσεων, η καθεμιά από τις οποίες ονομάζεται άτομο (individual) του πληθυσμού.

Βασικό χαρακτηριστικό του ΕΥ αποτελεί η «παράλληλη» αναζήτηση υποψηφίων λύσεων του προβλήματος στο χώρο αναζήτησης, καθώς οι υποψήφιες λύσεις του πληθυσμού δοκιμάζονται στο πρόβλημα «παράλληλα». Η παράλληλη αναζήτηση είναι πλήρως υλοποιήσιμη σε ένα υπολογιστικό περιβάλλον παράλληλης επεξεργασίας, Έτσι, σε αντίθεση με τους κλασικούς αλγορίθμους, όπου η βελτιστοποίηση μεθοδεύεται ψάχνοντας στο χώρο αναζήτησης σημείο-προς-σημείο, στον ΕΥ η βελτιστοποίηση μεθοδεύεται ψάχνοντας στο χώρο αναζήτησης ταυτόχρονα σε πολλά σημεία.

Με αυτόν τον τρόπο επιτυγχάνεται μια ευρύτερη εξερεύνηση (exploration), καθώς και μια στοχευμένη εκμετάλλευση (exploitation) του χώρου αναζήτησης προς εύρεση μιας βέλτιστης λύσης. Σε κάθε επανάληψη μεταξύ των ατόμων του πληθυσμού εφαρμόζονται κατάλληλοι τελεστές (operators), εμπνευσμένοι από τη Δαρβινική θεωρία της εξέλιξης, ώστε να παράγονται καλύτερες λύσεις σε κάθε επανάληψη. Τελικά, μετά από έναν αριθμό επαναλήψεων, η πειραματική εμπειρία έχει δείξει ότι ολόκληρος ο πληθυσμός συγκλίνει προς μία βέλτιστη λύση. Η εφαρμογή των τελεστών αποσκοπεί αφενός στην παραγωγή καλύτερων λύσεων και αφετέρου στην εξάλειψη χειρότερων λύσεων. Σε κάθε επανάληψη υπολογίζεται ένας νέος πληθυσμός ατόμων που περιέχει καλύτερες λύσεις με τελικό σκοπό να προσεγγιστεί το ολικό βέλτιστο του προβλήματος.

Ένα πρόβλημα το οποίο μπορεί να εμφανιστεί σε όλους τους αλγορίθμους ΕΥ είναι η πρόωρη σύγκλιση (premature conevrgence). Το φαινόμενο αυτό εμφανίζεται όταν ο αλγόριθμος προσεγγίζει πολύ γρήγορα μία λύση, δηλ. όταν όλες οι λύσεις του πληθυσμού αποκτούν την ίδια τιμή, οπότε τυπικά αποτρέπεται η δημιουργία διαφορετικών λύσεων

(μέσω της εφαρμογής των τελεστών) με αποτέλεσμα να μην εξερευνάται ευρέως ο χώρος αναζήτησης των λύσεων. Μία μεθόδευση για την αποφυγή πρόωρης σύγκλισης είναι η διατήρηση της ποικιλομορφίας (diversity) δηλ. μιας καλώς ορισμένης διαφορετικότητας των ατόμων του πληθυσμού ως αποτέλεσμα της εφαρμογής τελεστών.

2.3.2 Χαρακτηριστικοί Αλγόριθμοι

Δύο από τους δημοφιλέστερους αλγόριθμους ΕΥ είναι ο ΓΑ και ο αλγόριθμος ΒΣΣ. Ωστόσο, αξίζει να επισημανθεί ότι οι δύο αυτοί αλγόριθμοι αποτελούν ένα μικρό δείγμα εναλλακτικών αλγορίθμων ΕΥ από τους πολλούς που έχουν προταθεί στη βιβλιογραφία. Σημειώστε ότι όλοι οι αλγόριθμοι ΕΥ είναι ευρετικοί (heuristic) με την έννοια ότι εφαρμόζουν μία πειραματική διαδικασία αναζήτησης δοκιμή-και-σφάλμα (trial-and-error).

Ανάμεσα στο πλήθος των ευρετικών αλγορίθμων βελτιστοποίησης υπάρχουν τουλάχιστον δύο οι οποίοι παρουσιάζουν κοινά στοιχεία με αυτά των ΓΑ. Πρόκειται για τον αλγόριθμο της στρατηγικής εξέλιξης (Evolution Strategy) που είναι χρονολογικά προγενέστερος των ΓΑ και τον αλγόριθμο της διαφορικής εξέλιξης (Differential Evolution) που συχνά έχει βελτιωμένη απόδοση συγκριτικά με τους ΓΑ. Και οι δύο αυτοί αλγόριθμοι έχουν χρησιμοποιηθεί ευρύτατα για τη βελτιστοποίηση πολύπλοκων αντικειμενικών συναρτήσεων.

Δύο άλλοι ενδιαφέροντες αλγόριθμοι βελτιστοποίησης περιλαμβάνουν τη βελτιστοποίηση αποικίας μυρμηγκιών (Ant Colony) και την τεχνητή αποικία μελισσών (Artificial Bee Colony). Αυτοί ανήκουν σε μία ευρύτερη κατηγορία αλγορίθμων γνωστών με το όνομα νοημοσύνη σμήνους (Swarm Intelligence) και εξομοιώνουν τη συμπεριφορά σμηνών μυρμηγκιών και μελισσών για την εύρεση τροφής.

Αρκετά νεότερος από τους προαναφερθέντες δύο αλγορίθμους είναι ο αλγόριθμος βαρυτικής αναζήτησης (Gravitational Search), ο οποίος εξομοιώνει την έλξη μαζών σε βαρυτικό πεδίο και συνήθως έχει υψηλή απόδοση.

Επιπλέον διάφοροι αλγόριθμοι βελτιστοποίησης οι οποίοι προτάθηκαν και αναφέρονται στη βιβλιογραφία περιλαμβάνουν τον αλγόριθμο της πυγολαμπίδας (Firefly), τη μέθοδο αναζήτησης του κούκου (Cuckoo Search Method), τον αλγόριθμο της νυχτερίδας (Bat) και τη βελτιστοποίηση βιογεωγραφίας (Biogeography-Based Optimization).

Οι προαναφερθέντες αλγόριθμοι αποτελούν ένα (μικρό) υποσύνολο αλγόριθμων βελτιστοποίησης ΕΥ ή/και εμπνευσμένων από τη φύση που προτείνονται στη βιβλιογραφία.

Οι παραπάνω πληροφορίες για τον ΕΥ καθώς και η σχετική βιβλιογραφία υπάρχουν στο βιβλίο [14].

3. Η πορεία προς την Γραμματική Εξέλιξη

3.1 Γενετικοί Αλγόριθμοι

Ο ΓΑ επινοήθηκε από τον John Holland στα τέλη του 1960 στο Πανεπιστήμιο του Μίσιγκαν. Σήμερα μιλάμε, για Γενετικούς Αλγόριθμους και όχι για έναν Γενετικό Αλγόριθμο καθώς στην πορεία προτάθηκαν αρκετές διαφορετικές εκδόσεις του αρχικού Γενετικού Αλγόριθμου. Πρόκειται για αλγόριθμους που λύνουν μαθηματικά προβλήματα βελτιστοποίησης και ιδιαίτερα προβλήματα με πολλές παραμέτρους/διαστάσεις για τα οποία δεν υπάρχει γνωστή υπολογιστική μεθοδολογία για την επίλυση τους σε αποδεκτό χρόνο. [15]

3.1.1 Τρόπος λειτουργίας

Ένας ΓΑ δεν εγγυάται την εύρεση της βέλτιστης λύσης, αλλά μίας αρκούντως καλής λύσης. Για να φτάσει σε αυτή, χρησιμοποιεί έναν πληθυσμό που αποτελείται από κάποιον αριθμό ατόμων, όπου κάθε άτομο περιέχει σαν γενετική πληροφορία μία υποψήφια λύση του προβλήματος. Χρησιμοποιώντας κάποιους μηχανισμούς που συναντάμε στην βιολογία και πιό συγκεκριμένα στη θεωρία της εξέλιξης του Δαρβίνου, όπως η φυσική επιλογή, η διασταύρωση και η μετάλλαξη ο ΓΑ εξελίσσει τα άτομα του πληθυσμού.

Πιό συγκεκριμένα η δομή του βασικού ΓΑ αποτελείται από δύο τμήματα:

1. την αρχικοποίηση του πληθυσμού, η οποία αντιστοιχεί σε ένα τυχαίο σύνολο υποψήφιων λύσεων του προβλήματος, και
2. μία επανάληψη η οποία με τη σειρά της αποτελείται από:
    - την αξιολόγηση κάθε ατόμου του πληθυσμού και
    - την εφαρμογή των τελεστών της επιλογής, της διασταύρωσης και της μετάλλαξης.

Το αποτέλεσμα κάθε επανάληψης είναι μία καινούρια γενιά που αποτελεί εξέλιξη της προηγούμενης και από την οποία αναμένεται να βρεθεί μια βέλτιστη λύση. Ακολουθεί εκτενέστερη περιγραφή του τρόπου λειτουργίας του ΓΑ.

Αρχικοποίηση

Για την αρχικοποίηση του πληθυσμού πρέπει να καθοριστούν δύο παράμετροι: το πλήθος των στοιχείων του πληθυσμού, καθώς και ο τρόπος αναπαράστασης κάθε στοιχείου στον υπολογιστή (ή αλλιώς κωδικοποίηση).

Το κριτήριο με το οποίο επιλέγουμε τον τρόπο κωδικοποίησης μίας λύσης για ένα πρόβλημα, είναι ο τύπος του προβλήματος. Όταν λοιπόν πρόκειται για ένα πρόβλημα αριθμητικής βελτιστοποίησης τότε συνήθως επιλέγουμε δυαδική κωδικοποίηση, δηλαδή κάθε στοιχείο του πληθυσμού αναπαρίσταται με μία δυαδική συμβολοσειρά, ενώ στην περίπτωση που αντιμετωπίζουμε πρόβλημα συνδυαστικής βελτιστοποίησης τότε συνήθως προτιμάται η κωδικοποίηση ακεραίων.

Αξιολόγηση

Μετά την αρχικοποίηση του πληθυσμού ακολουθεί η βασική επανάληψη του αλγορίθμου. Το πρώτο βήμα της επανάληψης είναι η αξιολόγηση. Το σημαντικότερο ρόλο στο βήμα αυτό έχει η συνάρτηση ικανότητας (ή αξιολόγησης, ή καταλληλότητας, ή αντικειμενική συνάρτηση). Η συνάρτηση αυτή βαθμολογεί κάθε στοιχείο του πληθυσμού με έναν αριθμό που αναπαριστά το πόσο κοντά στην λύση του προβλήματος είναι το στοιχείο αυτό, σε σχέση με τα άλλα στοιχεία του πληθυσμού.

Επιλογή

Σε αυτό το βήμα της επανάληψης επιλέγεται ένα μέρος από τον τρέχοντα πληθυσμό, ανάλογα με τη βαθμολογία που έχει αποδοθεί σε κάθε στοιχείο από το βήμα της αξιολόγησης. Έτσι προκύπτει ο προσωρινός πληθυσμός στον οποίο θα εφαρμοστεί στη συνέχεια ο επόμενος τελεστής. Για την επιλογή χρειάζονται δύο παράμετροι. Η πρώτη καθορίζει το πλήθος των στοιχείων που θα επιλεχθούν και η δεύτερη τον τρόπο με τον οποίο θα γίνει η επιλογή.

Υπάρχουν διάφοροι τρόποι για την επιλογή των στοιχείων του πληθυσμού όπως της τυχαίας επιλογής, της εξαναγκασμένης ρουλέτας και άλλοι. Με βάση την φυσική επιλογή της θεωρίας του Δαρβίνου, ο τρόπος με τον οποίο επιλέγουμε ποιά στοιχεία θα αποτελέσουν τον προσωρινό πληθυσμό, θα πρέπει να επιτρέπει σε στοιχεία με υψηλή

βαθμολογία να υπάρχουν πάνω από μία φορά αλλά και να υπάρχουν στοιχεία με όχι και τόσο καλή βαθμολογία. Ο λόγος που θέλουμε ο προσωρινός πληθυσμός να έχει και στοιχεία με χαμηλή βαθμολογία είναι γιατί τέτοια στοιχεία μπορεί να έχουν κάποια χαρακτηριστικά που να αποτελούν μέρος της λύσης.

Διασταύρωση

Αφού προκύψει ο προσωρινός πληθυσμός επιλέγονται από αυτόν στοιχεία τα οποία διασταυρώνονται έτσι ώστε να προκύψουν νέα στοιχεία. Σε αυτό το βήμα πρέπει να καθοριστούν τρία πράγματα: α) το πλήθος των διασταυρώσεων που θα γίνουν, β) ποια στοιχεία του πληθυσμού θα συμμετέχουν σε κάθε διασταύρωση και γ) ο τρόπος με τον οποίο θα γίνει η διασταύρωση, (διασταύρωση ενός σημείου, διπλού σημείου κλπ). Ένα παράδειγμα διασταύρωσης ενός σημείου φαίνεται στην εικόνα που ακολουθεί.

Εικόνα 1 Διασταύρωση ενός σημείου στον Γενετικό Αλγόριθμο

Μετάλλαξη

Τέλος, η μετάλλαξη αλλοιώνει κάποια χαρακτηριστικά ορισμένων στοιχείων του πληθυσμού με τυχαίο τρόπο. Εδώ πρέπει πάλι να καθοριστούν δύο παράμετροι: ο τρόπος με τον οποίο θα επιλέξουμε τα στοιχεία και μία πιθανότητα που ονομάζεται πιθανότητα μετάλλαξης, η οποία χρησιμοποιείται στη λήψη της απόφασης για το αν θα αλλάξει ή όχι ένα χαρακτηριστικό κάποιου στοιχείου.

Τερματισμός Επανάληψης

Αφού εφαρμοστούν λοιπόν οι παραπάνω τελεστές προκύπτει ένας νέος πληθυσμός που αποτελεί την πρώτη γενιά και ξεκινά η επόμενη επανάληψη από την οποία θα προκύψει η δεύτερη γενιά κ.ο.κ. Η επανάληψη αυτή τερματίζει, είτε όταν φτάσουμε σε έναν αριθμό επαναλήψεων που έχουμε καθορίσει από την αρχή είτε αν για αρκετές γενιές δεν υπάρχει βελτίωση των στοιχείων του πληθυσμού. Όταν ολοκληρωθεί η επανάληψη κρατάμε την καλύτερη λύση που βρέθηκε από όλες τις γενιές και όχι μόνο από την τελευταία.

Τις παραπάνω πληροφορίες για τους ΓΑ, μπορεί να βρει κάποιος αναλυτικότερα στο [16].

3.1.2 Εφαρμογές

Ο ΓΑ πέρα από τα μαθηματικά προβλήματα έχει και πολλές άλλες εφαρμογές μερικές εκ των οποίων αναφέρονται στην συνέχεια.

Στον κλάδο της αυτοκινητοβιομηχανίας έχει εφαρμοστεί στη βελτιστοποίηση σχεδιασμού ενεργού συστήματος ανάρτησης οχημάτων [17] και παθητικής γραμμικής ανάρτησης [18], καθώς και στο βέλτιστο σχεδιασμό πολλαπλών αντικειμένων του κινητήρα αυτοκινήτων [19].

Σημαντικές εφαρμογές, ο ΓΑ έχει και στον τομέα της ρομποτικής, εκεί όπου χρησιμοποιείται κυρίως στον σχεδιασμό διαδρομής (Path Planning) των ρομπότ. Σε αυτόν το τομέα έχουν προταθεί διάφορες προσεγγίσεις του, όπως, παράλληλος, ελιτιστικός, βασισμένος στη γνώση καθώς και ο κλασσικός ΓΑ [20], [21], [22].

Ένας ακόμα τομέας που εφαρμόζεται ο ΓΑ είναι οι τηλεπικοινωνίες, στις οποίες χρησιμοποιείται κατά βάση για τη βελτιστοποίηση δρομολόγησης πακέτων. Περισσότερες πληροφορίες μπορεί να βρει κάποιος στο βιβλίο [23] στο οποίο αναλύονται διάφορες ευρετικές τεχνικές συμπεριλαμβανομένου του ΓΑ.

Στον τομέα των χρηματοοικονομικών και επενδυτικών στρατηγικών έχει χρησιμοποιηθεί μία προσέγγιση ΓΑ για την εξαγωγή επενδυτικών στρατηγικών με βάση τους κινητούς μέσους όρους [24] και για οικονομικές προβλέψεις [25]. Πολλές πληροφορίες για την εφαρμογή του ΓΑ σε επενδυτικές στρατηγικές υπάρχουν στο βιβλίο [26].

Τέλος έχει χρησιμοποιηθεί και σε ηλεκτρονικά παιχνίδια στρατηγικής [27], [28] αλλά και σε παιχνίδια First-Person Shooter [29].

3.2 Γενετικός Προγραμματισμός

Ο ΓΠ όπως τον ξέρουμε σήμερα, διατυπώθηκε για πρώτη φορά από τους Koza και de Garis το 1990. Σκοπός του ΓΠ δεν είναι να βρει την λύση ενός προβλήματος όπως ο ΓΑ, αλλά να παράξει ένα πρόγραμμα υπολογιστή, το οποίο με την σειρά του να επιλύει το αρχικό πρόβλημα. Παρόλα αυτά, ο τρόπος λειτουργίας του είναι παρόμοιος με εκείνον του ΓΑ, αφού χρησιμοποιούνται και πάλι μηχανισμοί της βιολογίας όπως η επιλογή, η διασταύρωση και η μετάλλαξη. Έτσι εξελίσεται ένας πληθυσμός προγραμμάτων υπολογιστή με σκοπό να βρεθεί το βέλτιστο πρόγραμμα με τη βοήθεια της αντικειμενικής συνάρτησης, που αξιολογεί κάθε ένα από τα υποψήφια προγράμματα.

Γενικά στον ΓΠ, οι υποψήφιες λύσεις αναπαρίστανται με δενδροηδή μορφή, για τις οποίες σημαντική παράμετρος είναι το πλήθος των κόμβων, το πλήθος των οποίων αναφέρεται και ως μέγεθος του ατόμου.

Τρόπος λειτουργίας

Πριν ξεκινήσει η εκτέλεση του βασικού αλγόριθμου, ο χρήστης πρέπει να ορίσει το πρόβλημα. Αυτό επιτυγχάνεται δίνοντας τα ακόλουθα στοιχεία :

- Το σύνολο των τερματικών στοιχείων, δηλαδή τις μεταβλητές, τις σταθερές και τις συναρτήσεις χωρίς είσοδο.
- Το σύνολο των αρχικών συναρτήσεων, δηλαδή τις εντολές, τους τελεστές και διάφορες συναρτήσεις που χρειάζεται ο αλγόριθμος για την παραγωγή των προγραμμάτων.
- Το μέτρο καταλληλότητας, δηλαδή την αντικειμενική συνάρτηση, η οποία αξιολογεί κατά πόσο μία λύση είναι καλύτερη σε σχέση με τις άλλες.
- Κάποιες άλλες παραμέτρους που αφορούν την εκτέλεση του αλγόριθμου όπως για παράδειγμα το μέγεθος του πληθυσμού, το μέγιστο πλήθος κόμβων κάθε ατόμου του πληθυσμού κ.λ.π.

- Το κριτήριο τερματισμού.

Ο βασικός αλγόριθμος

Αφού καθοριστούν τα παραπάνω στοιχεία από τον χρήστη ξεκινά η εκτέλεση του βασικού αλγόριθμου, η οποία όπως προαναφέρθηκε είναι παρόμοια με αυτήν των ΓΑ. Σε πρώτη φάση γίνεται τυχαία αρχικοποίηση του αρχικού πληθυσμού των ατόμων και ακολουθεί η βασική επανάληψη του αλγόριθμου. Σε αυτή εκτελούνται τα προγράμματα που αποτελούν τον πληθυσμό και αξιολογούνται με βάση την τιμή της αντικειμενικής συνάρτησης. Στη συνέχεια επιλέγονται ένα ή δυο άτομα του πληθυσμού στα οποία θα εφαρμοστούν οι τελεστές της αναπαραγωγής, της διασταύρωσης, της μετάλλαξης και των λειτουργιών μετατροπής αρχιτεκτονικής. Η επιλογή εξαρτάται από μία πιθανότητα η οποία βασίζεται στη βαθμολογία που έχει πάρει κάθε άτομο από την αντικειμενική συνάρτηση. Όταν ικανοποιηθεί η συνθήκη τερματισμού, το άτομο του τελικού πληθυσμού με την καλύτερη βαθμολογία αποτελεί την λύση, ή μία προσέγγιση της λύσης του προβλήματος.

Αρχικοποίηση

Για την αρχικοποίηση ενός πληθυσμού, χρειάζεται μία παράμετρος η οποία καθορίζει το μέγιστο μέγεθος κάθε ατόμου. Αν η αναπαράσταση των ατόμων γίνεται με δενδρική δομή, τότε η παράμετρος αυτή καθορίζει το μέγιστο πλήθος των κόμβων κάθε δέντρου. Υπάρχουν δύο τρόποι αρχικοποίησης.

Στον πρώτο τρόπο, οι κόμβοι παίρνουν τυχαία ένα στοιχείο από την ένωση των συνόλων των τερματικών και των συναρτήσεων, εκτός από τη ρίζα του δέντρου η οποία δεν μπορεί να περιέχει τερματικό άρα επιλέγεται τυχαία ένα στοιχείο από το σύνολο των συναρτήσεων. Αυτός ο τρόπος αρχικοποίησης παράγει δέντρα που δεν έχουν συνήθως κανονική μορφή.

Στο δεύτερο τρόπο, επιλέγονται τυχαία στοιχεία του συνόλου των συναρτήσεων μέχρι το δέντρο να φτάσει το μέγιστο βάθος και τότε επιλέγονται για φύλλα, τυχαία στοιχεία του συνόλου των τερματικών. Έτσι προκύπτουν ίδια δέντρα με το μέγιστο βάθος.

Αξιολόγηση

Μόλις ολοκληρωθεί η αρχικοποίηση του πληθυσμού ξεκινά η επανάληψη του αλγόριθμου που έχει σαν πρώτο βήμα την αξιολόγηση. Σε αυτό το βήμα εκτελούνται όλα τα προγράμματα του πληθυσμού και με την βοήθεια της αντικειμενικής συνάρτησης βαθμολογούνται με βάση την καταλληλότητά τους.

Επιλογή

Αφού καθοριστεί η ποιότητα μιας πιθανής λύσης από την αντικειμενική συνάρτηση, πρέπει να αποφασιστεί αν θα εφαρμοστούν οι γενετικοί τελεστές, σε αυτό το άτομο. Επίσης, από την απόδοσή του ατόμου εξαρτάται η διατήρησή του στον πληθυσμό, ή η αντικατάστασή του από κάποια άλλη λύση. Για αυτή τη διαδικασία, είναι υπεύθυνος ο τελεστής επιλογής. Υπάρχουν αρκετοί τέτοιοι τελεστές και η απόφαση για το ποιος θα χρησιμοποιηθεί, επηρεάζει σημαντικά την ταχύτητα της εξέλιξης. Τα σημαντικότερα είδη τελεστών επιλογής, είναι δύο:

1) Επιλογή ανάλογα με την απόδοση:

Αυτή η μέθοδος χρησιμοποιείται στους γενετικούς αλγορίθμους και είναι η πιο συνηθισμένη. Σύμφωνα με αυτήν, η πιθανότητα $\boldsymbol{p_i}$ να αφήσει απογόνους το άτομο $\boldsymbol{i}$ στην επόμενη γενιά, είναι:

$$\boldsymbol{p_i = f_i \Big/ \sum_{j=1}^{n} f_j}$$

2) Επιλογή με διαγωνισμό:

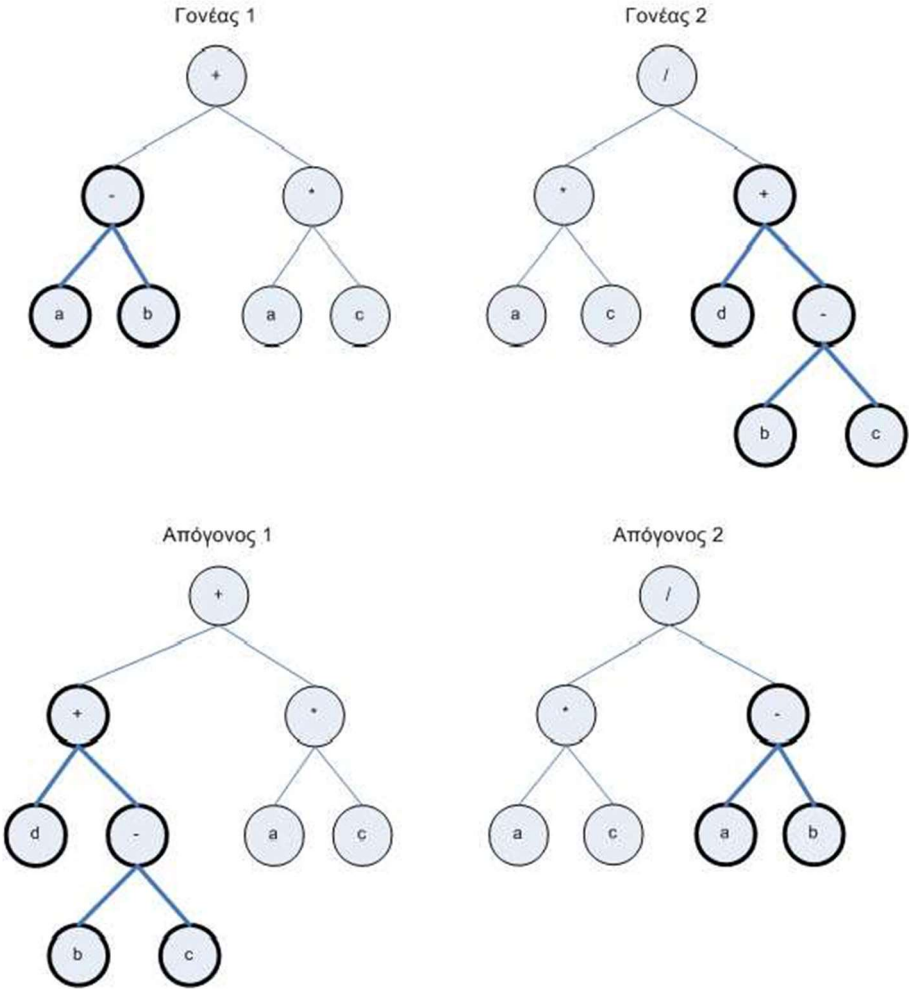
Αυτού του είδους η επιλογή βασίζεται στο διαγωνισμό με συγκεκριμένο αριθμό ατόμων και όχι με όλο τον πληθυσμό, όπως παραπάνω. Οι νικητές του διαγωνισμού αναπαράγονται χρησιμοποιώντας τους προηγούμενους τελεστές και οι απόγονοί τους αντικαθιστούν τους ηττημένους στον πληθυσμό. Αυτή η μορφή της επιλογής κερδίζει όλο και περισσότερο έδαφος, γιατί δεν απαιτεί κεντρικό υπολογισμό της απόδοσης κάθε ατόμου και επιταχύνει την εξέλιξη. Επίσης, είναι πολύ χρήσιμη, στην παράλληλη επεξεργασία.

Αναπαραγωγή

Ο τελεστής της αναπαραγωγής παίρνει τα άτομα του πληθυσμού που επιλέχθηκαν στο προηγούμενο βήμα και τα αντιγράφει αυτούσια στον καινούριο πληθυσμό.

Διασταύρωση

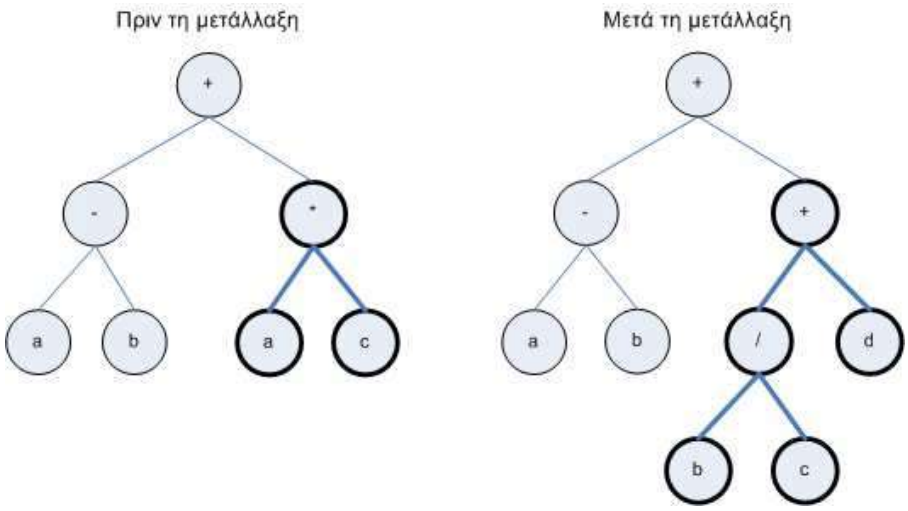
Μετά τον τελεστή της αναπαραγωγής ακολουθεί αυτός της διασταύρωσης. Σε αυτό το βήμα επιλέγονται δύο άτομα του πληθυσμού σύμφωνα με κάποιο κριτήριο. Ένας από τους συνηθέστερους τρόπους επιλογής είναι αυτός της εξαναγκασμένης ρουλέτας. Στην συνέχεια επιλέγεται τυχαία ένα υποδέντρο από κάθε γονέα. Τα υποδέντρα αυτά ανταλλάσσονται μεταξύ των γονέων και έτσι προκύπτουν δύο απόγονοι. Ένα παράδειγμα διασταύρωσης φαίνεται στην εικόνα που ακολουθεί.



Εικόνα 2 Διασταύρωση στον Γενετικό Προγραμματισμό

Μετάλλαξη

Τέλος, εφαρμόζεται ο τελεστής της μετάλλαξης. Για την εφαρμογή του απαιτείται μία πιθανότητα μετάλλαξης, όπως ονομάζεται, η οποία δίνεται από τον χρήστη σαν παράμετρος. Η πιθανότητα αυτή χρησιμοποιείται για να καθορίσει, ποιοί από τους απόγονους που προέκυψαν από τον τελεστή της διασταύρωσης, θα υποστούν μετάλλαξη. Για κάθε έναν από αυτούς επιλέγεται τυχαία ένα υποδέντρο του, το οποίο αντικαθίσταται από ένα άλλο υποδέντρο που παράγεται τυχαία. Ένα παράδειγμα μετάλλαξης φαίνεται στην παρακάτω εικόνα.



Εικόνα 3 Μετάλλαξη στον Γενετικό Προγραμματισμό

Τις παραπάνω πληροφορίες μπορεί να βρει κανείς αναλυτικότερα στο εκπαιδευτικό υλικό [30]

4. Γραμματική Εξέλιξη

Η ΓΕ είναι μία μορφή ΓΠ βασισμένη σε γραμματική. Συνδυάζει τις αρχές της μοριακής βιολογίας και των κανονικών γραμματικών, προσομοιώνοντας την πρωτεϊνοσύνθεση, δηλαδή τη διαδικασία μετάφρασης από τη γλώσσα των βάσεων στη γλώσσα των αμινοξέων, που συμβαίνει στο DNA ενός οργανισμού. Στη ΓΕ, τη διαδικασία αυτή, προσομοιώνει η λεγόμενη διαδικασία χαρτογράφησης γονότυπου σε φαινότυπο (genotype-to-phenotype mapping process).

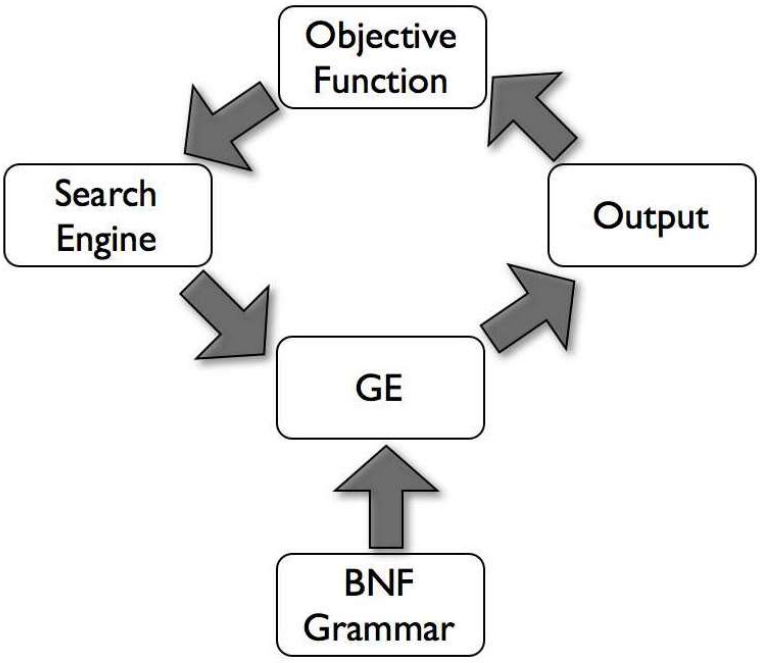
Πιο συγκεκριμένα, η διαδικασία χαρτογράφησης, ξεκινά από μία δυαδική συμβολοσειρά που ονομάζεται γονότυπος (genotype), η οποία αποτελεί το σύνολο των γονιδίων (genes) ενός οργανισμού, ή ενός ατόμου του πληθυσμού προς εξέλιξη, στην περίπτωση της ΓΕ. Στη συνέχεια, ο γονότυπος χωρίζεται σε διαδοχικές οκτάδες, και κάθε οκτάδα κωδικοποιεί έναν ακέραιο, ο οποίος ονομάζεται κωδικόνιο (codon), όπως αντίστοιχα συμβαίνει και στη βιολογία με τον κώδικα τριπλέτας. Τα κωδικόνια που προκύπτουν μετά την κωδικοποίηση τοποθετούνται σε μία συμβολοσειρά ακεραίων και στη συνέχεια χρησιμοποιούνται για την επιλογή κανόνων από τη δεδομένη BNF γραμματική (Backus Naur Form Grammar) με σκοπό την παραγωγή του φαινότυπου (phenotype), δηλαδή την παραγωγή ενός φαινοτυπικού προγράμματος.

Η επεκτασιμότητα της ΓΕ επιτρέπει την χρήση εναλλακτικών στρατηγικών αναζήτησης, όπως εξελικτικές, ή κάποιες άλλες ευρετικές (είτε στοχαστικές , είτε ντετερμινιστικές) και τη ριζική αλλαγής της συμπεριφορά της, απλά αλλάζοντας τη γραμματική που παρέχεται. Η ΓΕ έχει χρησιμοποιηθεί για την παραγωγή λύσεων σε πολλαπλές γλώσσες συμπεριλαμβανομένων των Lisp, Scheme, C/C++, Java, Prolog, Postscript, και στα Αγγλικά. Επίσης η διαδικασία χαρτογράφησης γονότυπου σε φαινότυπο σημαίνει ότι οι τελεστές αναζήτησης αντί να λειτουργούν αποκλειστικά σε δέντρα, όπως συμβαίνει στον ΓΠ, στη ΓΕ εκτελούνται στον γονότυπο (πχ ακέραιο ή δυαδικά χρωμοσώματα), επιπρόσθετα σε μερικώς παραγόμενους φαινοτύπους και ακόμα και στα πλήρως σχηματισμένα φαινοτυπικά δέντρα παραγωγής τα ίδια. Ως εκ τούτου, συνήθεις τελεστές δένδρων όπως αυτοί της διασταύρωσης υποδένδρου και μετάλλαξης υποδένδρου μπορούν εύκολα να υιοθετηθούν στην ΓΕ.

4.1 Τρόπος λειτουργίας

Σε αντίθεση με τον ΓΠ, η ΓΕ υιοθετεί έναν πληθυσμό από γραμμικές γονοτυπικές δυαδικές σειμβολοσειρές ή συμβολοσειρές ακεραίων, οι οποίες μετασχηματίζονται σε λειτουργικά φαινοτυπικά προγράμματα μέσω της διαδικασίας χαρτογράφησης γονότυπου σε φαινότυπο. Αυτός ο μετασχηματισμός διέπεται από τη χρήση μίας BNF γραμματικής, η οποία καθορίζει τη γλώσσα των παραγόμενων λύσεων.

Η ΓΕ είναι σπονδυλωτή στο σχεδιασμό της (αποτελείται από τμήματα), όπου η γραμματική, η μηχανή αναζήτησης και η αντικειμενική συνάρτηση, όλα αποτελούν εξαρτήματα της ΓΕ (Εικόνα 4). Αυτή η σπονδυλωτή μορφή, της επιτρέπει να χρησιμοποιείται σε συνδυασμό με αλγόριθμους αναζήτησης που προτιμάει κάθε ερευνητής. Σε πρόσφατες εξελίξεις, η ΓΕ, πέρα από τον κλασικό ΓΑ, έχει συνδυαστεί με αλγόριθμο σμήνους και με διαφοροεξελικτικό αλγόριθμο.



Εικόνα 4 Τρόπος λειτουργίας της Γραμματικής Εξέλιξης

Διαδικασία Χαρτογράφησης

Όταν ένα πρόβλημα προσεγγίζεται με ΓΕ, πρέπει αρχικά να οριστεί μία BNF γραμματική. Αυτή η γραμματική καθορίζει τη σύνταξη των επιθυμητών φαινοτυπικών προγραμμάτων, που θα παραχθούν από τη ΓΕ. Η σύνταξη ευρύτερα μπορεί να καθορίζει λεπτομερώς μία

γλώσσα προγραμματισμού όπως η C, ή καλύτερα ένα υποσύνολο μίας γλώσσας. Η ανάπτυξη μίας BNF γραμματικής παρέχει επίσης στον ερευνητή τη δυνατότητα να ενσωματώσει τμήματα της προτίμησης του ή τμήματα συγκεκριμένων συναρτήσεων.

Μία BNF γραμματική αποτελείται από μία τετράδα {N, T, P, S}, όπου N είναι το σύνολο όλων των μη τερματικών συμβόλων, T είναι το σύνολο των τερματικών, P είναι το σύνολο των κανόνων παραγωγής και S είναι το αρχικό σύμβολο και μέλος του N. Έστω λοιπόν, η ακόλουθη γραμματική:

N = {<expr>, <op>, <operand>, <var>}
T = {1, 2, 3, 4, +, -, /, *, x, y}
S = {<expr>}

τότε με P συμβολίζουμε τους κανόνες παραγωγής οι οποίοι σε BNF γράφονται ως εξής:

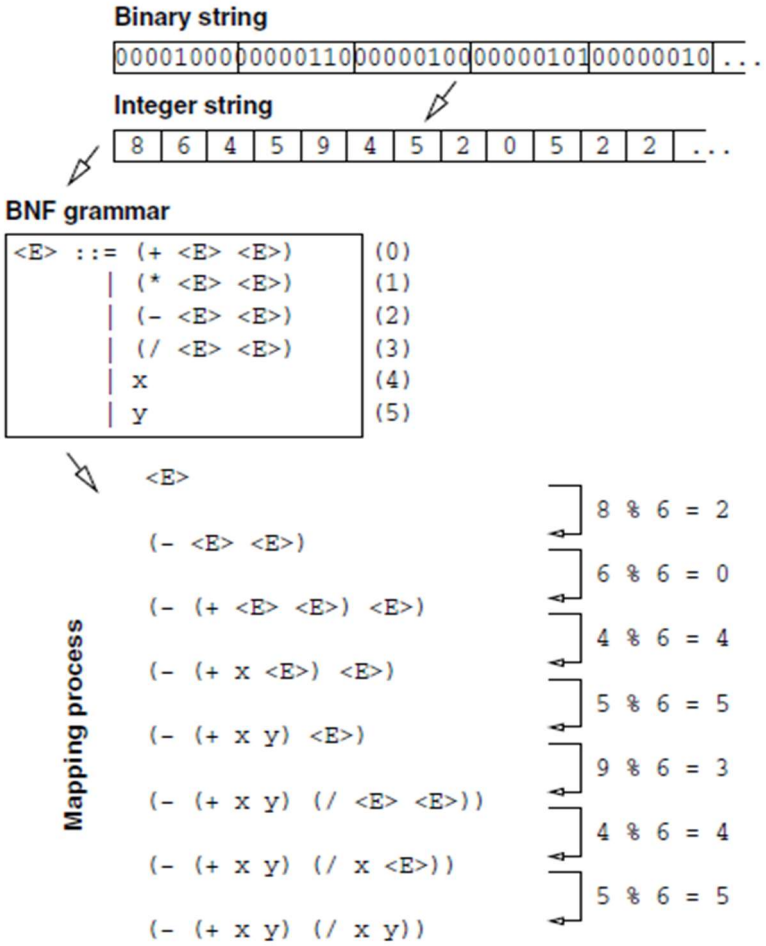
<expr>::= <expr><op><expr> **(0)**
| <operand> **(1)**
<op>::= + **(0)**
| - **(1)**
| / **(2)**
| * **(3)**
<operand>::= 1 **(0)**
| 2 **(1)**
| 3 **(2)**
| 4 **(3)**
| <var> **(4)**
<var>::= x **(0)**
| y **(1)**

Χρησιμοποιώντας μία τέτοια γραμματική για είσοδο, η ΓΕ στη συνέχεια εφαρμόζει την έκφραση :

$$\boldsymbol{Rule = c \,\%\, r}$$

με την οποία γίνεται η επιλογή του κανόνα παραγωγής για το σύμβολο που χαρτογραφείται επί του παρόντος, όπου ***c*** είναι η τιμή του κωδικόνιου και ***r*** ο αριθμός των διαθέσιμων κανόνων παραγωγής για το τρέχον μη τερματικό σύμβολο. Σημειώνεται ότι το σύμβολο ***"%"***, παριστάνει την πράξη modulo (δηλ. το υπόλοιπο της ακέραιας διαίρεσης).

Στην παρακάτω εικόνα φαίνεται ένα παράδειγμα της διαδικασίας χαρτογράφησης που χρησιμοποιείται από τη ΓΕ. Όπως αναφέρθηκε παραπάνω, ξεκινώντας από μία δυαδική συμβολοσειρά, προκύπτει μία συμβολοσειρά ακεραίων, τυπικά χρησιμοποιώντας 8 bits ανά κωδικόνιο, το οποίο είναι μία ακολουθία γονιδίων. Αυτά στη συνέχεια χρησιμοποιούνται για την επιλογή κανόνων από τη δεδομένη BNF γραμματική και για την παραγωγή ενός φαινοτυπικού προγράμματος.



Εικόνα 5 Διαδικασία χαρτογράφησης στην Γραμματικής Εξέλιξης

Ξεκινώντας με ένα δεδομένο αρχικό σύμβολο ***<E>***, επιλέγεται μία παραγωγή που συνδέεται με αυτό το σύμβολο για να το αντικαταστήσει, χρησιμοποιώντας το τρέχον κωδικόνιο από τη συμβολοσειρά ακεραίων. Στο παράδειγμα, το κωδικόνιο 8 χαρτογραφείται στον αριθμό των διαθέσιμων παραγωγών που σχετίζονται με το ***<E>***, για το οποίο υπάρχουν 6 παραγωγές. Επομένως 8 mod 6 = 2, και το ***<E>*** αντικαθίσταται από την ακολουθία ***(- <E><E>)***.

Το επόμενο βήμα αποτελείται από την επιλογή μίας παραγωγής για το επόμενο μη τερματικό σύμβολο, δηλαδή το ***<E>***, το οποίο είναι τώρα το πιο αριστερό σύμβολο της υπό κατασκευή συμβολοσειράς φαινοτύπου. Αυτό γίνεται με τη χρήση του επόμενου κωδικόνιου 4, το οποίο πάλι χαρτογραφείται στον αριθμό παραγωγών που σχετίζονται με το ***<E>***, δίνοντας 4 mod 6 = 4, οπότε το ***<E>*** αντικαθίσταται με ***(+ <E><E>)***.

Η διαδικασία χαρτογράφησης συνεχίζεται με αυτόν τον τρόπο, αντικαθιστώντας πάντα το πιο αριστερό μη τερματικό σύμβολο, με μια παραγωγή που σχετίζεται με αυτό το σύμβολο στη γραμματική, το οποίο επιλέγεται από ένα κωδικόνιο. Η χαρτογράφηση στην ΓΕ ολοκληρώνεται όταν ικανοποιηθεί μία από τις παρακάτω συνθήκες :

- Δημιουργείται ένα πλήρες πρόγραμμα πριν από το τέλος του γονιδιώματος. Αυτό συμβαίνει όταν όλα τα μη τερματικά σύμβολα της έκφρασης η οποία χαρτογραφείται μετασχηματιστούν σε στοιχεία από το σύνολο των τερματικών της BNF γραμματικής όπως φαίνεται στο παράδειγμα της εικόνας 5.
- Φτάνει στο τέλος του γονιδιώματος, όπου σε αυτή την περίπτωση εφαρμόζεται ο τελεστής αναδίπλωσης. Αυτό έχει ως αποτέλεσμα την επαναφορά του πλαισίου ανάγνωσης του γονιδιώματος στην αριστερή πλευρά του γονιδιώματος για άλλη μια φορά. Η ανάγνωση των κωδικονίων θα συνεχιστεί όπως πριν, εκτός εάν επιτευχθεί ένα όριο, το οποίο αντιπροσωπεύει τον μέγιστο αριθμό αναδιπλώσεων, κατά τη διάρκεια της διαδικασίας της χαρτογράφησης.
- Φτάνει στο μέγιστο όριο αναδίπλωσης και το άτομο δεν έχει ακόμα χαρτογραφηθεί πλήρως. Σε αυτή την περίπτωση η διαδικασία χαρτογράφησης σταματάει και ανατίθεται στο άτομο η χειρότερη δυνατή τιμή ικανότητας.

Μετάλλαξη και Διασταύρωση

Όπως ήδη περιγράφηκε, η αναπαράσταση του ατόμου στο οποίο εφαρμόζονται οι γενετικοί τελεστές στη ΓΕ είναι γραμμικές συμβολοσειρές μεταβλητού μεγέθους. Λόγω της αναπαράστασης αυτής, οι μηχανισμοί των τελεστών του κλασικού ΓΑ παραμένουν ίδιοι, η μετάλλαξη αλλάζει ένα bit ή έναν ακέραιο σε μία διαφορετική τιμή και η διασταύρωση μονού σημείου αντιμεταθέτει κομμάτια του γενετικού κώδικα μεταξύ γονέων. Ωστόσο, λόγω της διαδικασίας χαρτογράφησης, η επίδραση στο φαινότυπο μπορεί να είναι περίπλοκη.

Στη ΓΕ και στο ΓΑ του Holland, ο τελεστής μετάλλαξης εφαρμόζεται χωρίς περιορισμούς. Ωστόσο, στη ΓΕ υπάρχει το ενδεχόμενο, η μετάλλαξη, να μην έχει αποτέλεσμα ή να είναι ουδέτερη στο φαινοτυπικό επίπεδο. Για παράδειγμα, έστω ο ακόλουθος κανόνας παραγωγής BNF :

<var>::= x (0)
| y (1)

και μία ακέραια τιμή κωδικόνιου 20. Το μη τερματικό ***<var>*** θα αντικατασταθεί από την μεταβλητή ***x*** επειδή, χρησιμοποιώντας τον κανόνα χαρτογράφησης που αναφέρθηκε προηγουμένως, ο κανόνας που θα εκτελεστεί είναι 20 % 2, που έχει σαν αποτέλεσμα 0, ή τον πρώτο κανόνα παραγωγής. Αν προκύψει μία μετάλλαξη και αλλάξει το κωδικόνιο σε 21, το ***<var>*** θα αντικατασταθεί με ***y***. Ωστόσο αν αυτό το κωδικόνιο αλλάξει σε 22, ή οποιονδήποτε άλλο άρτιο αριθμό στη συγκεκριμένη περίπτωση, (εφόσον υπάρχουν μόνο δύο κανόνες παραγωγής) ο φαινότυπος θα παραμείνει ***x***. Αυτό σημαίνει ότι συμβαίνει ουδέτερη μετάλλαξη καθώς η λειτουργικότητα του εκφραζόμενου γονιδίου που προκύπτει παραμένει η ίδια.

Η διασταύρωση ενός σημείου στη ΓΕ γίνεται με τον ίδιο τρόπο όπως και στο ΓΑ, όπου το γενετικό υλικό στα δεξιά των επιλεγμένων σημείων διασταύρωσης, αντιμετατίθεται μεταξύ δύο γονέων. Στην περίπτωση της ΓΕ, και πάλι λόγω της διαδικασίας χαρτογράφησης, το αποτέλεσμα στο φαινοτυπικό επίπεδο είναι διαφορετικό από το αντίστοιχο στο ΓΑ. Η διασταύρωση στη ΓΕ έχει ως επίδραση το φαινόμενο εξάπλωσης (ripple effect) στην ακολουθία εφαρμογής των κανόνων παραγωγής μετά το σημείο διασταύρωσης. Στον πυρήνα της διασταύρωσης της ΓΕ ισχύει η εξής αρχή: όταν το γενετικό υλικό ανταλλάσσεται με άλλο άτομο, το πλαίσιο μέσα στο οποίο τοποθετείται μπορεί να είναι διαφορετικό. Αυτό έχει ως αποτέλεσμα μια διαφορετική φαινοτυπική έκφραση σε σύγκριση με όταν ήταν στην αρχική τοποθέτησή της. Ενώ αυτή η μορφή διασταύρωσης μπορεί να φαίνεται καταστροφική, η έρευνα έχει δείξει ότι μεταφέρονται χρήσιμες αλληλουχίες παραγωγής. Η διαδικασία χαρτογράφησης στη ΓΕ μπορεί να περιγραφεί ως μια ακολουθία παραγωγών, η οποία με τη σειρά της μπορεί να αναπαρασταθεί ως δέντρο παραγωγής.

4.2 Εφαρμογές

Όπως αναφέρεται στο υποκεφάλαιο 2.5 του βιβλίου [31], από την αρχική της διατύπωση και εμφάνιση η ΓΕ έχει προσελκύσει το ερευνητικό ενδιαφέρον και έχει εφαρμοστεί σε πολλά και διαφορετικά ήδη προβλημάτων. Πιό συγκεκριμένα εφαρμόστηκε με επιτυχία σε προβλήματα συμβολικής παλινδρόμησης (symbolic regression), στην εξέλιξη των τριγωνομετρικών ταυτοτήτων καθώς και στην εξέλιξη των αλγορίθμων κρυπτογράφησης και ρομποτικής συμπεριφοράς.

Η ΓΕ επεκτάθηκε και εφαρμόστηκε στον τομέα του σχεδιασμού επιφάνειας (domain of surface design), όπου συνδυάστηκε με το GENR8, ένα εργαλείο σχεδιασμού επιφάνειας που χρησιμοποιεί μια εξελικτική διαδικασία για την προσαρμογή των επιφανειών ώστε να πληροί τις προδιαγραφές του χρήστη.

Στον τομέα της βιοπληροφορικής, η ΓΕ εφαρμόστηκε για να αναγνωρίσει ευκαρυωτικούς προαγωγούς, που βοηθούν στην αναγνώριση των βιολογικών γονιδίων. Χρησιμοποιήθηκε επίσης για την ανάπτυξη νευρωνικών δικτύων για επιλογή χαρακτηριστικών στη γενετική επιδημιολογία.

Στον τομέα της ηχητικής σύνθεσης και ανάλυσης, χρησιμοποιήθηκε για την αυτόματη δημιουργία συνθέσεων που ήταν συγκρίσιμες με ανθρώπινες συνθέσεις. Επίσης εφαρμόστηκε για την εξέλιξη των φωνολογικών κανόνων που μπορούν να χρησιμοποιηθούν για τη μετάφραση του κειμένου σε γραφήματα και την αναγνώριση των ήχων ως λέξεων.

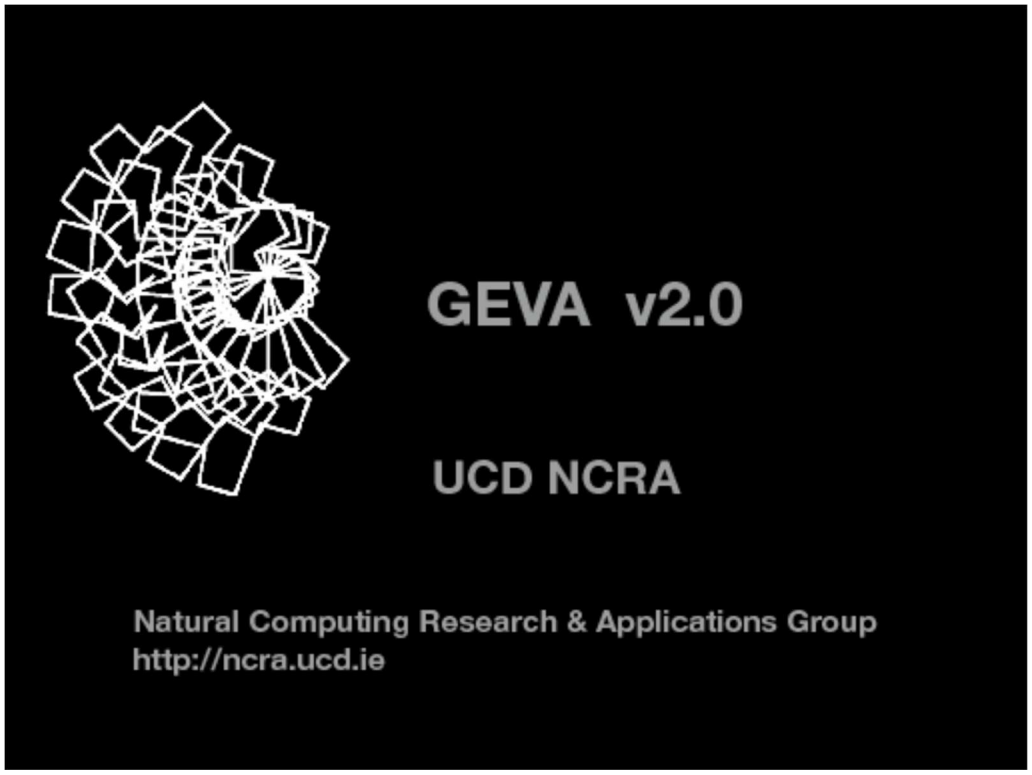
Τέλος, η ΓΕ έχει εφαρμοστεί και σε διάφορους τομείς της οικονομίας σε μελέτες συναλλαγών σε ξένο συνάλλαγμα, σε πτώχευση και εταιρική ανάλυση, καθώς και σε προβλήματα πιστοληπτικής διαβάθμισης.

Οι παραπάνω πληροφορίες για τη ΓΕ υπάρχουν στο βιβλίο [31] όπου μπορεί κανείς να βρει περισσότερες λεπτομέρειες για τη ΓΕ και για τις εφαρμογές της, καθώς και σχετικές βιβλιογραφικές αναφορές.

5. Υλοποίηση και αποτελέσματα

5.1 Περιγραφή του GEVA

Το περιβάλλον Γραμματικής Εξέλιξης σε Java GEVA (Grammatical Evolution in Java), αναπτύχθηκε στο Natural Computing Research & Applications group του University College Dublin. Το περιβάλλον αποτελεί μία εφαρμογή ΓΕ ανοιχτού κώδικα, που διατίθεται βάσει του GNU GPL v3.0, και προσφέρει ένα πλαίσιο μηχανής αναζήτησης, μία απλή γραφική διεπαφή χρήστη (GUI) και το πρόγραμμα αντιστοίχισης γονότυπου-φαινοτύπου της ΓΕ.



Εικόνα 6 Η αρχική οθόνη του GEVA

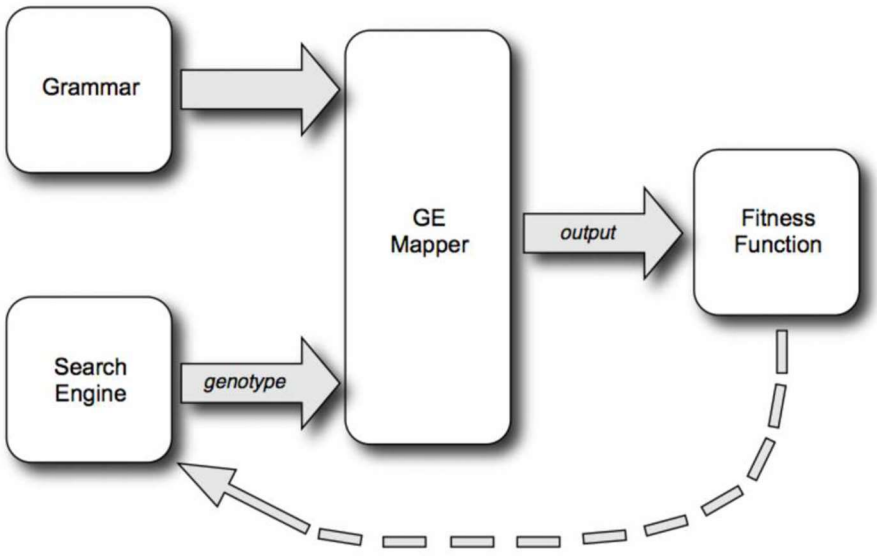
Το λογισμικό αποτελείται από δύο βασικά μέρη, μία απλή γραφική διεπαφή χρήστη και το κυρίως πακέτο GEVA. Το γραφικό περιβάλλον παρέχει ένα μηχανισμό για την εξερεύνηση του λογισμικού και την κατανόηση του τρόπου λειτουργίας του, αλλά η χρήση του GEVA μπορεί να γίνει μέσω μίας διεπαφής γραμμής εντολών για να επιτρέψει τη

δημιουργία σεναρίων. Σε αυτή την έκδοση (v2.0) παρέχονται κάποια απλά προβλήματα επίδειξης για να βοηθήσουν το χρήστη στην κατανόηση του κώδικα ενώ διαβάζει το συνοδευτικό διδακτικό υλικό που παρέχεται στον οδηγό του περιβάλλοντος και στη σελίδα του κώδικα.

Για να μπορέσουμε να επιλύσουμε οποιοδήποτε πρόβλημα με το GEVA, πρέπει να καθορίσουμε τα ακόλουθα :

- τη BNF γραμματική η οποία χρησιμοποιείται για την παραγωγή των υποψήφιων λύσεων,
- την αντικειμενική συνάρτηση η οποία αξιολογεί κάθε φορά μία λύση σύμφωνα με τους κανόνες και το στόχο κάθε προβλήματος, και
- τις παραμέτρους του ΓΑ, όπως για παράδειγμα το πλήθος των ατόμων στον πληθυσμό, τον μέγιστο αριθμό γενεών, τις πιθανότητες διασταύρωσης και μετάλλαξης κ.α.

Ο τρόπος που αυτά τα στοιχεία συνδυάζονται φαίνεται στην παρακάτω εικόνα.



Εικόνα 7 Συστατικά στοιχεία της Γραμματικής Εξέλιξης

Για να εισάγουμε λοιπόν, ένα πρόβλημα στο GEVA πρέπει πρώτα να καθορίσουμε τη γραμματική. Για να γίνει αυτό πρέπει να δημιουργήσουμε ένα *.bnf* αρχείο που θα περιέχει

τους κανόνες παραγωγής της γραμματικής σε Backus-Naur form και να το αποθηκεύσουμε στο μονοπάτι *"<GEVA_ROOT>/param/Grammar/"*.

Στη συνέχεια πρέπει να καθορίσουμε την αντικειμενική συνάρτηση του προβλήματος. Μία αντικειμενική συνάρτηση στο GEVA γράφεται σε Java και πρέπει να περιέχεται μέσα σε μία κλάση η οποία με τη σειρά της πρέπει να επεκτείνει την κλάση FitnessFunction. Αφού δημιουργήσουμε το αρχείο *.java* που περιέχει την κλάση, πρέπει να το τοποθετήσουμε σε έναν φάκελο στη διαδρομή *"<GEVA_ROOT>/GEVA/src/FitnessEvaluation/"* εκεί όπου το GEVA διατηρεί όλες τις αντικειμενικές συναρτήσεις.

Τέλος πρέπει να καθορίσουμε το αρχείο με τις παραμέτρους. Το αρχείο αυτό στο GEVA αποθηκεύεται με την κατάληξη *.properties* και πέρα από τις παραμέτρους του κλασσικού ΓΑ μπορεί να περιέχει και άλλες παραμέτρους που χρειάζεται να χρησιμοποιήσουμε ανάλογα με το πρόβλημα που προσπαθούμε να λύσουμε. Όλα τα αρχεία παραμέτρων αποθηκεύονται στην διαδρομή *"<GEVA_ROOT>/param/Parameters/"*.

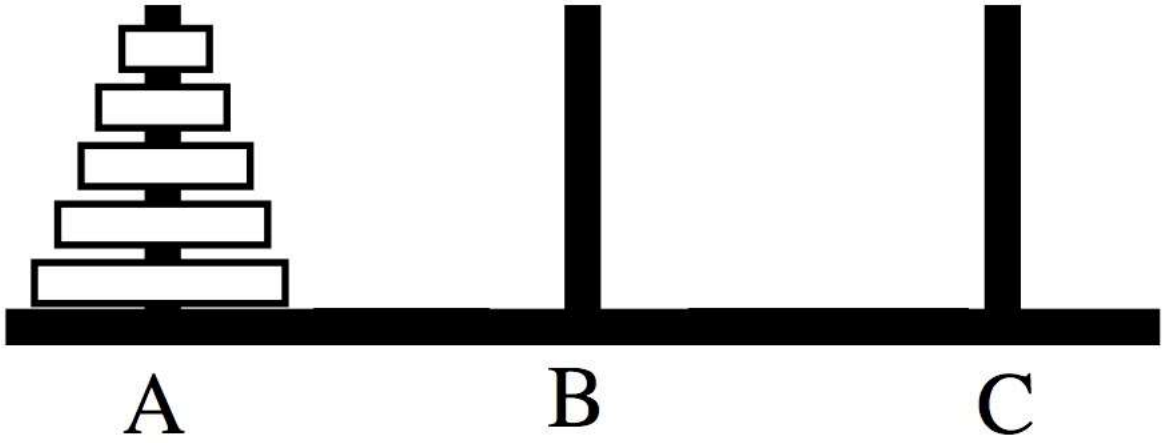
Μερικές παράμετροι που αξίζει να σημειωθούν είναι :

- *mutation_probability* (πιθανότητα μετάλλαξης): καθορίζει πόσο πιθανό είναι να μεταλλαχθεί κάθε άτομο του πληθυσμού
- *initializer* (αρχικοποιητής): καθορίζει τον τρόπο αρχικοποίησης του πληθυσμού
- *generations* (γενιές): καθορίζει το μέγιστο πλήθος γενεών που θα εκτελεστεί ο αλγόριθμος
- *crossover_operation* (διαδικασία διασταύρωσης): καθορίζει τον τρόπο με τον οποίο τα άτομα διασταυρώνονται.
- *population_size* (μέγεθος πληθυσμού): καθορίζει το μέγεθος του πληθυσμού που θα εξελιχθεί
- *crossover_probability* (πιθανότητα διασταύρωσης): καθορίζει πόσο πιθανό είναι να διασταυρωθούν δύο άτομα του πληθυσμού
- *mutation_operation* (διαδικασία μετάλλαξης): καθορίζει τον τρόπο με τον οποίο θα γίνεται η μετάλλαξη στα άτομα του πληθυσμού

Αναλυτική περιγραφή των παραπάνω πληροφοριών καθώς και περισσότερες λεπτομέρειες για το GEVA και τον τρόπο χρήσης του παρέχονται στο [32].

5.2 Οι Πύργοι του Ανόι

Το πρόβλημα των Πύργων του Ανόι παρουσιάστηκε για πρώτη φορά από τον Γάλλο μαθηματικό Edouard Lucas το 1883. Αποτελείται από μία βάση πάνω στην οποία υπάρχουν στερεωμένοι τρεις στύλοι. Στον πρώτο στύλο υπάρχουν τοποθετημένοι δίσκοι διαφορετικής διαμέτρου. Οι δίσκοι είναι τοποθετημένοι έτσι ώστε στη βάση του στύλου να βρίσκεται ο δίσκος με τη μεγαλύτερη διάμετρο και οι υπόλοιποι διατάσσονται κατά φθίνουσα σειρά από τη μεγαλύτερη διάμετρο στη μικρότερη όπως φαίνεται στην παρακάτω εικόνα.



Εικόνα 8 Οι Πύργοι του Ανόι

Σκοπός είναι να μετακινηθούν όλοι οι δίσκοι από τον πρώτο στύλο στον τρίτο χωρίς να παραβιαστούν οι δύο ακόλουθοι περιορισμοί: α) δεν επιτρέπεται να μετακινηθεί πάνω από ένας δίσκος καθε φορά και β) δεν μπορεί να τοποθετηθεί μεγαλύτερος δίσκος πάνω σε μικρότερο.

Οι Πύργοι του Ανόι είναι το πρώτο πρόβλημα που επιλέξαμε, καθώς είναι ένα πολύ γνωστό και απλό στην αναπαράσταση του πρόβλημα. Το πρόβλημα αυτό αν και δεν αποτελεί ένα παιχνίδι γρίφου, εν τούτοις χρησιμοποιήθηκε, κυρίως για την κατανόηση και εξοικείωση με το περιβάλλον GEVA, καθώς και για την εξάσκηση στη χρήση γραμματικών και παραμέτρων. Στην πραγματικότητα είναι ένα διακριτό (ντετερμινιστικό) πρόβλημα, δηλαδή απαιτεί έναν συγκεκριμένο αλγόριθμο για την επίλυση του και άρα δεν έχει το στοιχείο της βελτιστοποίησης που μας ενδιαφέρει. Παρόλα αυτά είναι ένα

πρόβλημα με πολλές εκδοχές και άρα παρουσιάζει ενδιαφέρον για την εξερεύνηση των διαφορετικών πιθανών εκδοχών με τη ΓΕ.

BNF γραμματική

Η γραμματική που χρησιμοποιήσαμε για το πρόβλημα των Πύργων του Ανόι είναι η ακόλουθη :

<s> ::= <from>,<to>;|<from>,<to>;<s>
<from> ::= 1|2|3
<to> ::= 1|2|3

Από την γραμματική φαίνεται ότι ξεκινώντας από την αρχική κατάσταση ***<s>*** μπορεί να προκύψει είτε μία κίνηση, δηλαδή ένα ζευγάρι ***<from><to>*** όπου το ***<from>*** δείχνει από ποιον στύλο παίρνουμε τον δίσκο και το ***<to>*** σε ποιον στύλο τον τοποθετούμε, είτε μία ακολουθία τέτοιων κινήσεων.

Αντικειμενική Συνάρτηση

Ο κώδικας της κλάσης η οποία περιέχει την αντικειμενική συνάρτηση που δημιουργήσαμε για την αξιολόγηση των υποψήφιων λύσεων βρίσκεται στο Παράρτημα 1.

Η λογική που ακολουθήσαμε είναι η ακόλουθη: η αντικειμενική συνάρτηση δέχεται μία υποψήφια λύση η οποία όπως είπαμε προηγουμένως αποτελείται από μία ή περισσότερες μετακινήσεις δίσκων. Εκτελεί με τη σειρά όλες τις κινήσεις παραλείποντας αυτές που δεν είναι έγκυρες. Όταν δεν υπάρχουν άλλες κινήσεις αξιολογεί την κατάσταση που έχει καταλήξει ως εξής: κάθε στύλος όπως και κάθε δίσκος έχει ένα βάρος. Συγκεκριμένα στους στύλους, εφόσον ξεκινάμε από τον πρώτο και πρέπει να καταλήξουμε στον τρίτο δίνουμε για βάρος την τιμή 2 στον πρώτο στύλο, την τιμή 1 στον δεύτερο στύλο και την τιμή 0 στον τρίτο. Σε ότι αφορά τους δίσκους εφόσον ο μικρότερος δίσκος είναι ο πιο εύκολος στην μετακίνηση και όσο μεγαλώνουν οι δίσκοι τόσο δυσκολεύει η μετακίνηση τους δίνουμε για βάρος την τιμή 1 στον μικρότερο, την τιμή 2 στον επόμενο, την τιμή 3 στον επόμενο, κ.ο.κ.. Κατά συνέπεια για ένα πρόβλημα με N δίσκους ο μεγαλύτερος δίσκος θα έχει βάρος N. Επίσης ρόλο παίζει και ο αριθμός των μη επιτρεπτών κινήσεων.

Έτσι η τιμή της αντικειμενικής συνάρτησης για μία υποψήφια λύση προκύπτει από τον τύπο:

$$\boldsymbol{Fitness} = \boldsymbol{em} + \sum_{\boldsymbol{i=1}}^{\boldsymbol{disks}} (\boldsymbol{dw(i)} \cdot \boldsymbol{cw})$$

όπου ***disks*** είναι ο αριθμός των δίσκων, **dw(i)** (disk weight) το βάρος του ***i*** δίσκου, ***cw*** (column weight) το βάρος του στύλου στον οποίον βρίσκεται ο δίσκος και ***em*** (error moves) το πλήθος των μη επιτρεπτών κινήσεων. Δηλαδή, η τιμή της αντικειμενικής συνάρτησης υπολογίζεται από το άθροισμα των γινομένων του βάρους κάθε δίσκου επί το βάρος του στύλου στον οποίο καταλήγει να βρίσκεται συν το πλήθος τον μη επιτρεπτών κινήσεων. Στόχος του προβλήματος βελτιστοποίησης είναι η ελαχιστοποίηση της αντικειμενικής συνάρτησης. Η βέλτιστη λύση είναι αυτή που θα αντιστοιχεί στη στρατηγική επίλυσης η οποία θα έχει μεταθέσει όλους τους δίσκους από τον πρώτο στύλο στον τρίτο και θα έχει το μικρότερο αριθμό λανθασμένων κινήσεων (***em)***.

Τα αποτελέσματα του GEVA σε σύγκριση με τα αποτελέσματα που επιτυγχάνει ο κλασικός τρόπος επίλυσης του συγκεκριμένου προβλήματος μας έδειξαν ότι η χρήση της ΓΕ δεν είναι ίσως η καλύτερη μέθοδος για την επίλυση του προβλήματος αυτού.

5.3 Το Mastermind

Το σύγχρονο παιχνίδι επινοήθηκε το 1970 από τον Mordecai Meirowitz, ισραηλινό ειδήμονα και ειδικό στον τομέα των τηλεπικοινωνιών. Το Mastermind είναι ένα παιχνίδι μυαλού και παίζεται με δύο παίκτες.

Ο πρώτος παίκτης δημιουργεί έναν τετραψήφιο κωδικό έχοντας στη διάθεση του 8 διαφορετικά χρώματα. Ο κωδικός αυτός μπορεί να περιέχει περισσότερες από μία φορές το ίδιο χρώμα. Ο δεύτερος παίκτης καλείται να βρει τον τετραψήφιο αυτό κωδικό χρωμάτων μέσα σε 10 προσπάθειες.

Σε κάθε προσπάθεια ο δεύτερος παίκτης προτείνει έναν συνδυασμό χρωμάτων και στη συνέχεια ο πρώτος, του λέει πόσα από τα χρώματα που περιέχει ο κωδικός που πρότεινε υπάρχουν όντως στον κωδικό και πόσα από αυτά τα χρώματα είναι τοποθετημένα στη σωστή θέση. Στη συνέχεια ο δεύτερος κάνει μία καινούρια υπόθεση προτείνοντας πάλι

έναν τετραψήφιο αριθμό και ο πρώτος του λέει πάλι τις παραπάνω πληροφορίες για την καινούρια υπόθεσή του.

Η διαδικασία αυτή σταματάει όταν ο δεύτερος παίκτης είτε ολοκληρώσει την $10^{η}$ προσπάθεια χωρίς να καταφέρει να βρει τον κωδικό, είτε καταφέρει σε κάποια προσπάθεια του μέσα στις 10 επιτρεπόμενες να βρει τον κωδικό.

5.3.1 Εισαγωγή του Mastermind στο GEVA

Όπως αναφέραμε προηγουμένως για να εισάγουμε ένα πρόβλημα στο περιβάλλον GEVA πρέπει να καθορίσουμε την γραμματική BNF, την αντικειμενική συνάρτηση, καθώς και το αρχείο παραμέτρων.

BNF γραμματική

Η γραμματική που χρησιμοποιήσαμε για το πρόβλημα του Mastermind είναι η ακόλουθη:

<pattern> ::= <guess>|<guess>;<pattern>
<guess> ::= <col><col><col><col>
<col> ::= 1|2|3|4|5|6|7|8

Βλέπουμε από την γραμματική ότι ξεκινώντας από την αρχική κατάσταση ***<pattern>*** μπορεί να προκύψει είτε μία υπόθεση, είτε μία ακολουθία υποθέσεων με οποιοδήποτε μέγεθος οι οποίες αναπαριστούν τις προσπάθειες του παίκτη να βρει τον κρυμμένο κωδικό. Κάθε υπόθεση αποτελείται από τέσσερα διαδοχικά μη τερματικά σύμβολα ***<col>*** τα οποία αναπαριστούν ένα χρώμα και μπορούν να αντικατασταθούν με έναν ακέραιο αριθμό από το πεδίο [1,8] (ένας αριθμός για κάθε διαφορετικό χρώμα).

Αντικειμενική συνάρτηση

Μετά τον καθορισμό της BNF γραμματικής , πρέπει να καθορίστεί η αντικειμενική συνάρτηση. Ο κώδικας της κλάσης που περιέχει την αντικειμενική συνάρτηση που χρησιμοποιήσαμε για την αξιολόγηση των λύσεων υπάρχει στο Παράρτημα 2.

Η συνάρτηση Java, που υλοποιεί την αντικειμενική συνάρτηση, δέχεται τις υποψήφιες λύσεις που έχουν παραχθεί από την BNF γραμματική. Για κάθε μία από αυτές εφαρμόζει την ακόλουθη διαδικασία :

Χωρίζει την υποψήφια λύση σε τετράδες (υποθέσεις) και αξιολογεί την κάθε υπόθεση της λύσης με τον παρακάτω τρόπο :

- Δίνει στην υπόθεση ως αρχική ικανότητα, την τιμή 60.0
- Για κάθε χρώμα της υπόθεσης που υπάρχει στη λύση αφαιρεί από την ικανότητά της 5 ενώ για κάθε χρώμα που βρίσκεται στη σωστή θέση αφαιρεί 10.
- Έτσι, εάν η υπόθεση ταυτίζεται με τη λύση όλα τα χρώματα της υπόθεσης θα υπάρχουν στη λύση άρα θα αφαιρεθεί από την αρχική της ικανότητα 5 για κάθε χρώμα δηλαδή 4 x 5 = 20. Αν όλα τα χρώματα είναι τοποθετημένα στην σωστή θέση τότε θα αφαιρεθεί επιπρόσθετα από την ικανότητά της 10 για κάθε χρώμα δηλαδή 4 x 10 = 40. Άρα θα μηδενιστεί η ικανότητα της υπόθεσης που σημαίνει ότι είναι σωστή.

Είναι προφανές όμως, ότι μία υποψήφια λύση που βρίσκει τελικά τον κωδικό και αποτελείται από τρεις για παράδειγμα υποθέσεις, είναι χειρότερη από μία άλλη που επίσης βρίσκει τον κωδικό αλλά αποτελείται είτε από δύο είτε από μία υπόθεση.

Για τον λόγο αυτό η αντικειμενική συνάρτηση επιστρέφει μηδέν, και κατά συνέπεια, σταματάει η εκτέλεση του αλγόριθμου, μόνον όταν βρεθεί υποψήφια λύση η οποία είναι σωστή και αποτελείται από μία και μοναδική υπόθεση. Σε διαφορετική περίπτωση η υποψήφια λύση αξιολογείται ως εξής :

- Αν η λύση περιέχει δέκα ή λιγότερες υποθέσεις, εφόσον υπάρχει μέγιστο όριο δέκα υποθέσεων, τότε παίρνει σαν ικανότητα το πλήθος των υποθέσεων που περιέχει συν την ικανότητα που δόθηκε στην τελευταία της υπόθεση.
- Αν η λύση περιέχει πάνω από δέκα υποθέσεις, τότε παίρνει για ικανότητα 9 για τις πρώτες εννέα υποθέσεις της συν τις ικανότητες που έχουν δοθεί σε κάθε μία από τις υποθέσεις της, από την δέκατη και μετά.

Παράμετροι Γενετικού Αλγόριθμου

Παρακάτω φαίνονται οι παράμετροι που χρησιμοποιήθηκαν για μία εκτέλεση του προγράμματος για το συγκεκριμένο πρόβλημα και για έναν συγκεκριμένο κωδικό. Δεν υπάρχει λόγος να παρουσιάσουμε ξεχωριστά τις παραμέτρους που χρησιμοποιήθηκαν για κάθε διαφορετικό κωδικό, καθώς το μόνο που αλλάζει κάθε φορά είναι η παράμετρος κωδικός.

```
1.  main_class=Main.Run
2.  mutation_probability=0.02
3.  initialiser=Operator.RampedHalfAndHalfInitialiser
4.  grow_probability = 0.5
5.  derivation_tree=Mapper.DerivationTree
6.  max_depth = 10
7.  tail_percentage=0
8.  generations=100
9.  replacement_type=generational
10. generation_gap=0.5
11. crossover_operation=Operator.Operations.SinglePointCrossover
12. fixed_point_crossover=false
13. evaluate_elites=false
14. userpick_size=20
15. grammar_file=../param/Grammar/Mastermind.bnf
16. population_size=20
17. output=
18. fitness_function=FitnessEvaluation.Mastermind.Mastermind
19. max_wraps=0
20. selection_operation=Operator.Operations.TournamentSelect
21. crossover_probability=0.7
22. mutation_operation=Operator.Operations.IntFlipMutation
23. elite_size=10
24. tournament_size=3
25. stopWhenSolved=true
26. code = 4444
27. tail_percentage = 0.5
```

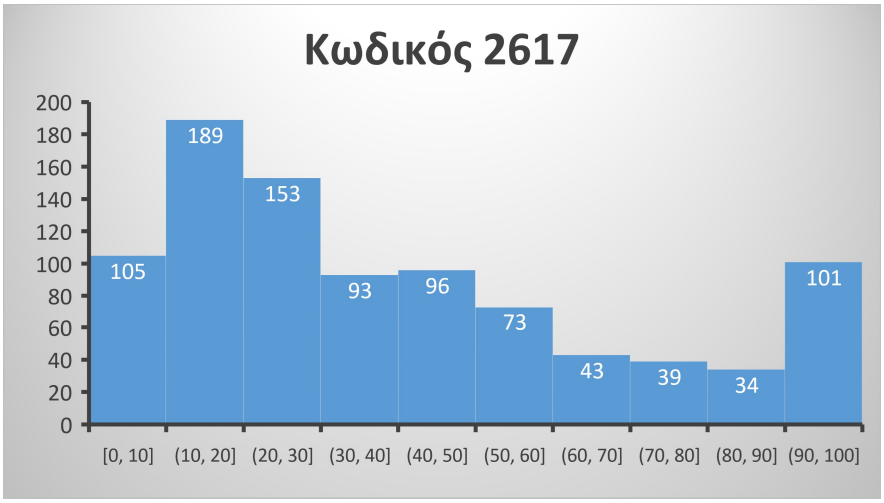
5.3.2 Πειραματική αξιολόγηση

Περιγραφή του πειράματος

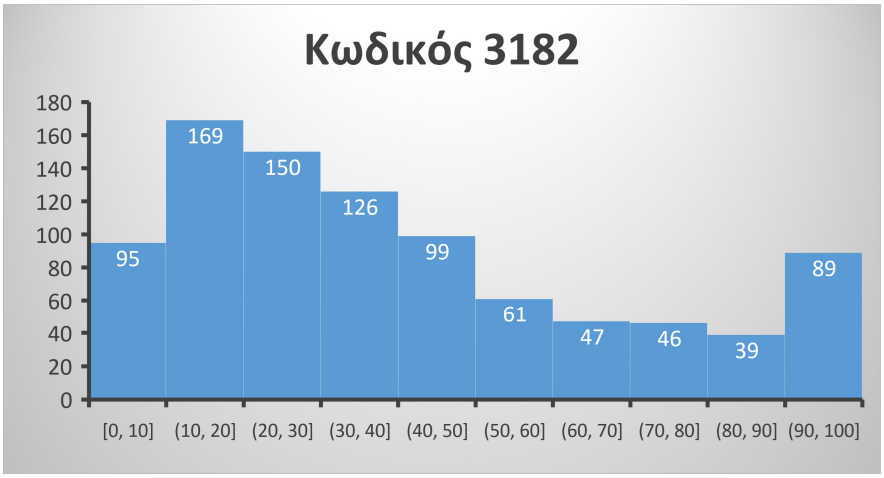
Η αξιολόγηση της αποδοτικότητας του αλγόριθμου της ΓΕ ως προς την ικανότητά του να επιλύει το πρόβλημα, έγινε επιλέγοντας τυχαία 5 διαφορετικούς κωδικούς, τους εξής: 2617, 3182, 4441, 5337, 8171. Για κάθε έναν από αυτούς τους κωδικούς εκτελέσαμε τον αλγόριθμο χίλιες φορές με μέγιστο αριθμό γενεών 100 κάνοντας χρήση ενός Batch file, ο κώδικας του οποίου φαίνεται στο Παράρτημα 3. Τα αποτελέσματα των πειραμάτων καταγράφηκαν σε αρχεία.

Πειραματικά Αποτελέσματα

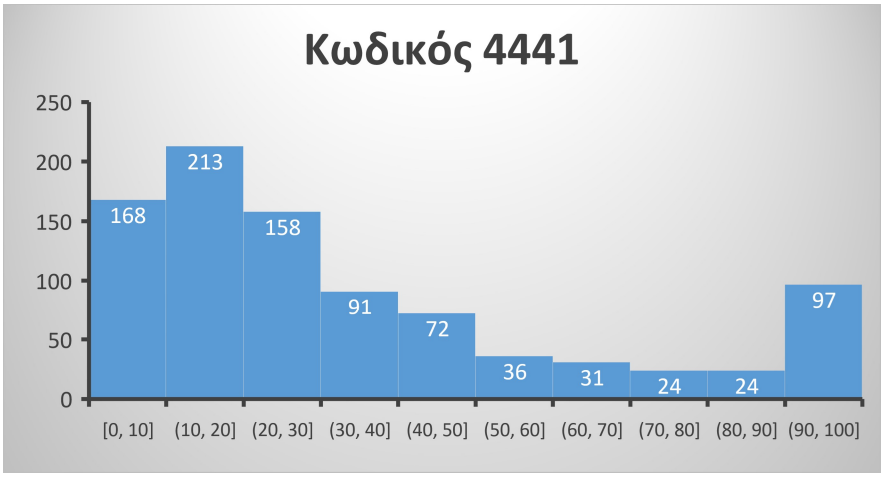
Στη συνέχεια, και μόνο για τις εκτελέσεις στις οποίες ο αλγόριθμος βρήκε την λύση, υπολογίσαμε, για κάθε ξεχωριστό κωδικό ένα ιστόγραμμα που παρουσιάζει τον αριθμό των γενεών που χρειάστηκε κάθε εκτέλεση του αλγόριθμου, για να φτάσει στη λύση. Τα ιστογράμματα αυτά φαίνονται στη συνέχεια.



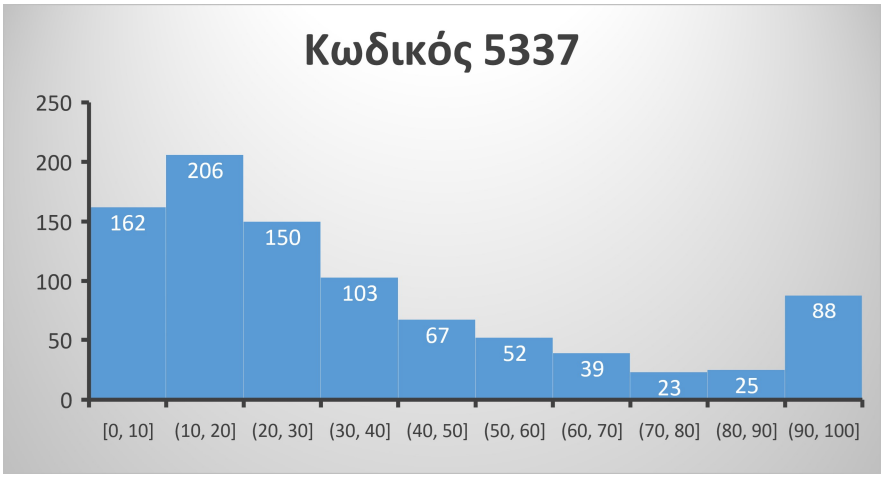
Εικόνα 9 Ιστόγραμμα του Mastermind για κωδικό 2617



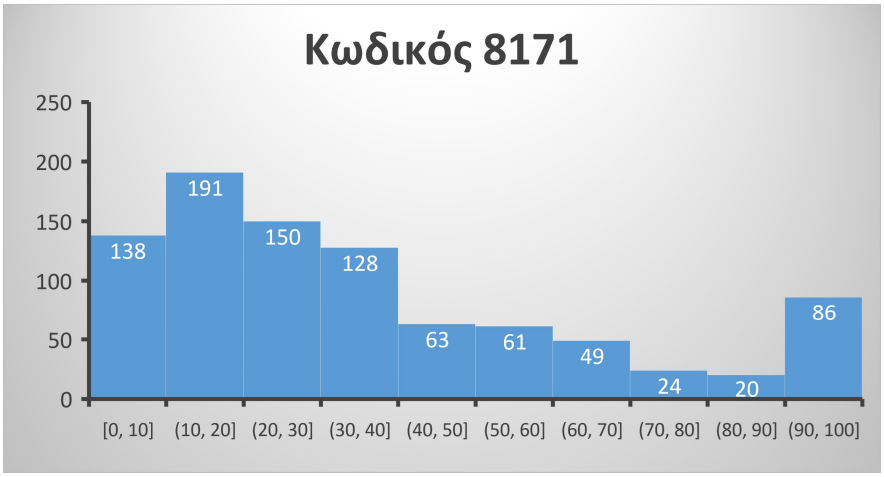
Εικόνα 10 Ιστόγραμμα του Mastermind για κωδικό 3182



Εικόνα 11 Ιστόγραμμα του Mastermind για κωδικό 4441



Εικόνα 12 Ιστόγραμμα του Mastermind για κωδικό 5337



Εικόνα 13 Ιστόγραμμα του Mastermind για κωδικό 8171

Για να χαρακτηρίσουμε την κεντρική τάση των αποτελεσμάτων υπολογίσαμε τις τιμές των εξής δειγματικών στατιστικών:

- Μέση τιμή (mean)
- Διάμεσος (median)

Ως μέτρο διασποράς των αποτελεσμάτων υπολογίσαμε τη δειγματική τυπική απόκλιση (standard deviation). Για τον υπολογισμό των στατιστικών αυτών, δημιουργήσαμε μία κλάση σε Java η οποία φαίνεται στο Παράρτημα 4. Τα αποτελέσματα των υπολογισμών δίνονται συνοπτικά στον παρακάτω πίνακα.

Κωδικοί	***Ποσοστό Επιτυχίας***	***Διάμεσος***	***Μέσος***	***Τυπική Απόκλιση***
2617	92.6 %	32	40.64	29.15
3182	92.1 %	33	41.12	28.37
4441	91.4 %	25	35.42	29.48
5337	91.5 %	26	35.66	28.73
8171	91 %	29	37.09	28.11

Πίνακας 1 Ανάλυση αποτελεσμάτων του Mastermind

Συμπέρασμα

Από τη μορφή των ιστογραμμάτων συχνότητας που στην πλειονότητά τους είναι δεξιά ή αλλιώς θετικά ασύμμετρα (right skewed) φαίνεται ότι ο μεγαλύτερος όγκος των αποτελεσμάτων βρίσκεται σε μικρότερες τιμές του αριθμού των γενεών. Οι τιμή της μέσης τιμής για κάθε πείραμα δίνει μια πρώτη προσέγγιση της κεντρικής θέσης των αποτελεσμάτων αν και λόγω της μορφής των ιστογραμμάτων οι τιμές των διαμέσων δείχνουν να είναι πιό αντιπροσωπευτικές για την κεντρική τάση των πειραματικών αποτελσμάτων.

Βλέπουμε λοιπόν από τις τιμές της διάμεσου σε κάθε πείραμα ότι στις περισσότερες εκτελέσεις του αλγόριθμου, η λύση βρέθηκε σε μικρό αριθμό γενεών άρα και σε μικρό χρονικό διάστημα.

5.4 Το SameGame

Το SameGame κυκλοφόρησε αρχικά με το όνομα Chain Shot! το 1985 από τον Kuniaki Moribe (Morisuke).

Αποτελείται από έναν δισδιάστατο πίνακα, κάθε θέση του οποίου έχει ένα χρώμα. Το πλήθος των διαφορετικών χρωμάτων μέσα στον πίνακα καθορίζει συνήθως το επίπεδο δυσκολίας. Στο εύκολο επίπεδο ο πίνακας αποτελείται από τρία διαφορετικά χρώματα, στο μέτριο επίπεδο από τέσσερα, ενώ στο δύσκολο από πέντε. Επιλέγοντας μία ομάδα γειτονικών χρωμάτων, ο παίκτης μπορεί να αφαιρέσει την ομάδα αυτή από τον πίνακα. Κάθε φορά που αφαιρείται από τον πίνακα μία ομάδα, τα χρώματα που δεν στηρίζονται πλέον πάνω σε άλλα πέφτουν προς τα κάτω και όταν αδειάσει τελείως μία στήλη τότε όλες στήλες που βρίσκονται δεξιά της μετακινούνται κατά μία θέση αριστερά.

Υπάρχουν δύο διαφορετικές εκδοχές για το στόχο του παιχνιδιού. Στην πρώτη εκδοχή ο στόχος είναι να μείνουν στον πίνακα όσο το δυνατόν λιγότερα χρώματα, ενώ στην άλλη ο στόχος είναι ο παίκτης να πετύχει τη μεγαλύτερη δυνατή βαθμολογία. Η τελική βαθμολογία προκύπτει από το άθροισμα των βαθμολογιών κάθε κίνησης του παίκτη, όπου η βαθμολογία κάθε ξεχωριστής κίνησης προκύπτει συνήθως από τον παρακάτω τύπο:

$$\boldsymbol{rate = (n - k)^2}$$

όπου $\boldsymbol{n}$ το πλήθος των γειτονικών χρωμάτων που αφαιρέθηκαν κατά την κίνηση αυτή, και $\boldsymbol{k}$ = 1 ή 2 ανάλογα με την υλοποίηση.

Το παιχνίδι ολοκληρώνεται όταν ο παίκτης δεν μπορεί να κάνει άλλη κίνηση, δηλαδή δεν υπάρχουν γειτονικά χρώματα για να αφαιρεθούν.

5.4.1 Εισαγωγή του SameGame στο GEVA

BNF γραμματική

Η γραμματική που χρησιμοποιήσαμε για το πρόβλημα του SameGame φαίνεται παρακάτω:

<pattern> ::= <num><num>|<num><num>;<pattern>

<num> ::= 0|1|2|3|4|5|6|7|8

Μία κίνηση στο SameGame είναι η επιλογή μίας θέσης του πίνακα. Έτσι στη γραμματική ξεκινώντας από την αρχική κατάσταση ***<pattern>*** μπορεί να προκύψει είτε μία κίνηση, είτε μία ακολουθία κινήσεων. Κάθε κίνηση αναπαριστάται με ένα ζευγάρι ***<num><num>***, όπου το ***<num>*** παίρνει ακέραιες τιμές στο πεδίο [0,8], αφού ο πίνακας που χρησιμοποιούμε έχει μέγεθος 9x9.

Αντικειμενική συνάρτηση

Όπως αναφέρθηκε προηγουμένως, υπάρχουν δύο διαφορετικές εκδοχές του παιχνιδιού ως προς το στόχο. Στα πλαίσια της πτυχιακής επιλέξαμε την εκδοχή όπου ο στόχος είναι να ελαχιστοποιηθεί ο αριθμός των χρωμάτων που θα μείνει στο τέλος, δηλαδή όταν δε θα μπορεί να πραγματοποιηθεί άλλη κίνηση. Ο κώδικας Java, της κλάσης η οποία περιλαμβάνει την αντικειμενική συνάρτηση φαίνεται στο Παράρτημα 5.

Ο τρόπος λειτουργίας της αντικειμενικής συνάρτησης είναι απλός. Αρχικά διαβάζει τον πίνακα χρωμάτων από ένα αρχείο το οποίο αποτελείται από τυχαίους αριθμούς ανάλογα με τη δυσκολία του προβλήματος. Για το εύκολο επίπεδο όπου έχουμε τρία διαφορετικά χρώματα, το αρχείο αποτελείται από τυχαίους ακεραίους αριθμούς στο διάστημα [1,3] τοποθετημένους σε εννέα γραμμές και εννέα στήλες, ενώ για το μεσαίο επίπεδο το αρχείο αποτελείται από τυχαίους ακεραίους στο διάστημα [1,4]. Στη συνέχεια, εφόσον η αντικειμενική συνάρτηση γεμίσει τον πίνακα χρωμάτων, δέχεται μία ακολουθία κινήσεων και τις εφαρμόζει στον πίνακα. Τέλος, αφού εφαρμοστούν όλες οι κινήσεις μίας υποψήφιας λύσης η αντικειμενική συνάρτηση επιστρέφει τον αριθμό των χρωμάτων που έχουν μείνει στον πίνακα. Προφανώς το καλύτερο σενάριο είναι να μη μείνει κανένα χρώμα μετά την εφαρμογή των κινήσεων δηλαδή η αντικειμενική συνάρτηση να επιστρέψει την τιμή μηδέν.

Παράμετροι Γενετικού Αλγόριθμου

Οι τιμές των βασικών παραμέτρων είναι οι ίδιες με αυτές που χρησιμοποιήσαμε και παραπάνω στο πρόβλημα του Mastermind. Η μόνη διαφορά που υπάρχει στο αρχείο των παραμέτρων είναι ότι σε αυτό το πρόβλημα δε δίνουμε σαν παράμετρο τον κωδικό αλλά

το όνομα του αρχείου που θέλουμε να χρησιμοποιήσουμε για να γεμίσουμε τον πίνακα χρωμάτων.

5.4.2 Πειραματική αξιολόγηση

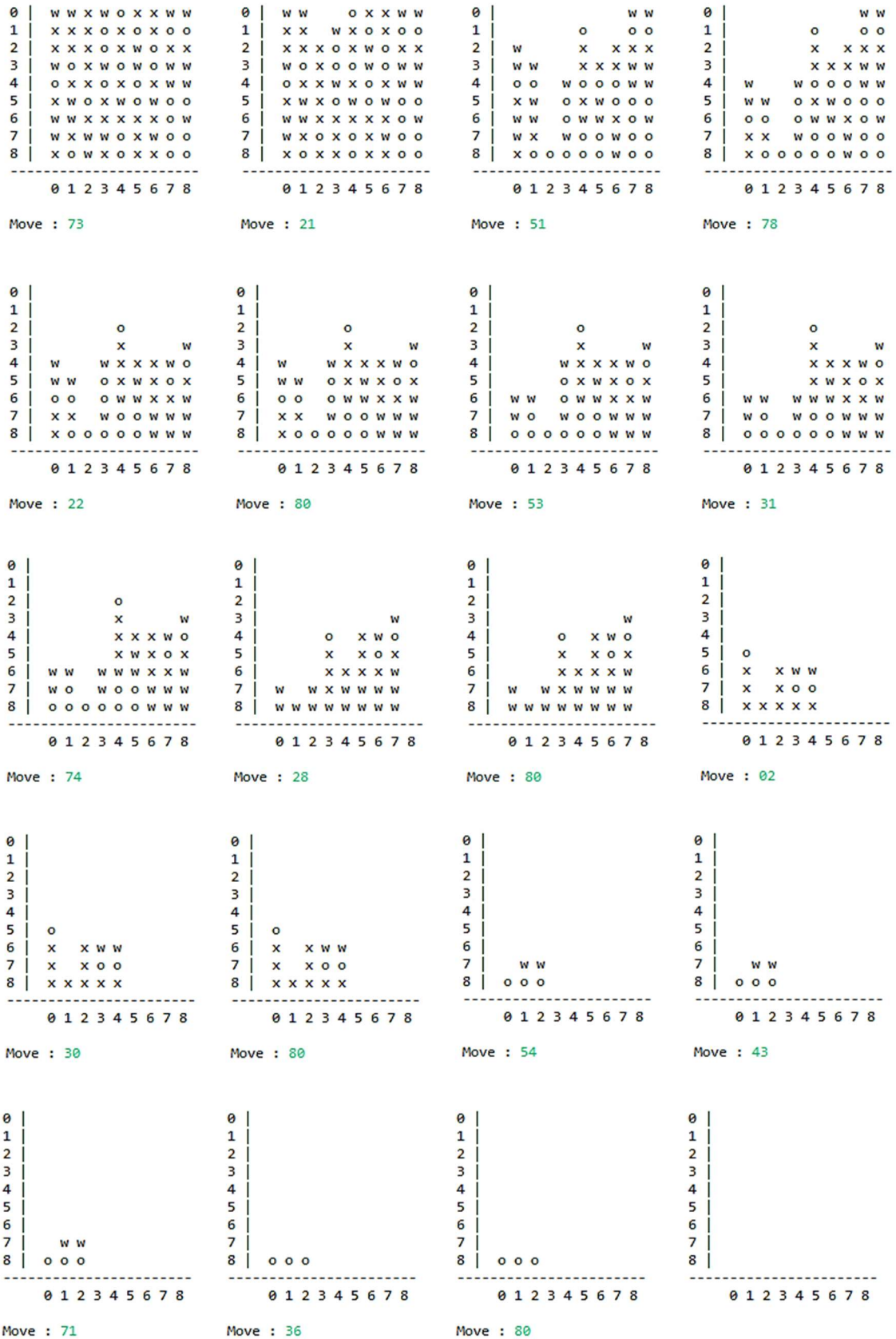
Περιγραφή του πειράματος

Δημιουργήσαμε έξι διαφορετικά αρχεία, τρία για το εύκολο και άλλα τρία για το μεσαίο επίπεδο. Κάθε ένα από αυτά αποτελείται από ογδόντα ένα τυχαίους ακέραιους αριθμούς στο πεδίο [1,3] για το εύκολο επίπεδο και στο [1,4] για το μεσαίο επίπεδο έτσι ώστε να αναπαραστήσουμε τα τρία και τα τέσσερα διαφορετικά χρώματα του εύκολου και του μεσαίου επίπεδου αντίστοιχα. Οι τυχαίοι αριθμοί μέσα σε κάθε αρχείο είναι διατεταγμένοι σε εννέα γραμμές και εννέα στήλες.

Εκτελέσαμε τον αλγόριθμο χίλιες φορές για κάθε διαφορετικό αρχείο, με μέγιστο αριθμό γενεών 100 για τα αρχεία του εύκολου και 500 για αυτά του δύσκολου επίπεδου, έτσι ώστε να ελέγξουμε τη συμπεριφορά του, κάνοντας κάποιες μικρές αλλαγές στο σχετικό Batch file που δημιουργήσαμε όπως και στο προηγούμενο πρόβλημα (Mastermind).

Για την επαλήθευση των λύσεων αλλά και για την καλύτερη κατανόηση του προβλήματος δημιουργήσαμε ένα πρόγραμμα σε Java που μιμείται τη λειτουργία του παιχνιδιού SameGame. Γεμίζει τον πίνακα χρωμάτων από ένα αρχείο κειμένου και προβάλει τον πίνακα με μορφή συμβόλων. Ο χρήστης εισάγει κάθε φορά μία κίνηση δίνοντας τις συντεταγμένες μίας θέσης του πίνακα μέχρι να μην μπορεί να εκτελεστεί άλλη κίνηση. Ο κώδικάς του προγράμματος αυτού υπάρχει στο Παράρτημα 6.

Στην εικόνα 14, φαίνεται ένα παράδειγμα εκτέλεσης του προγράμματος που περιγράψαμε προηγουμένως, εφαρμόζοντας μία από τις λύσεις που βρήκαμε μέσω του GEVA για την επίλυση του SameGame, χρησιμοποιώντας το πρώτο αρχείο εύκολου επίπεδου. Ο φαινότυπος, δηλαδή η λύση που προέκυψε από το GEVA είναι : 7 3 ; 2 1 ; 5 1 ; 7 8 ; 2 2 ; 8 0 ; 5 3 ; 3 1 ; 7 4 ; 2 8 ; 8 0 ; 0 2 ; 3 0 ; 8 0 ; 5 4 ; 4 3 ; 7 1 ; 3 6 ; 8 0 ; 7 7 ; 7 4 ; 3 2

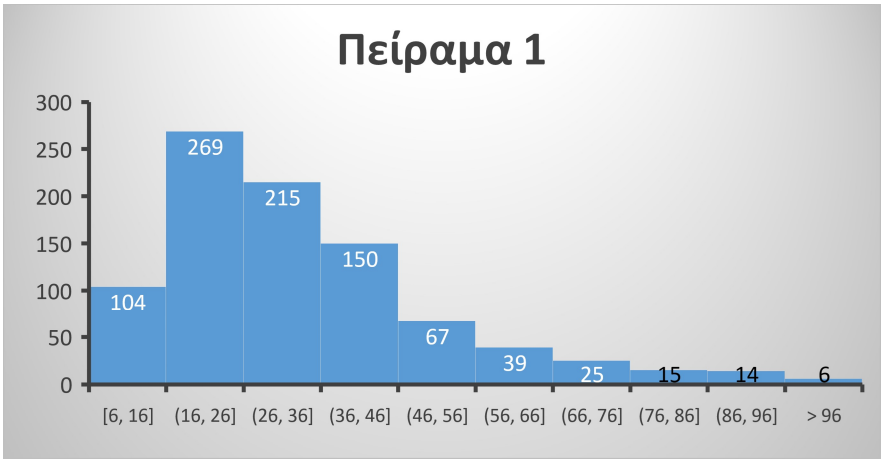


Εικόνα 14 Παράδειγμα εκτέλεσης του SameGame

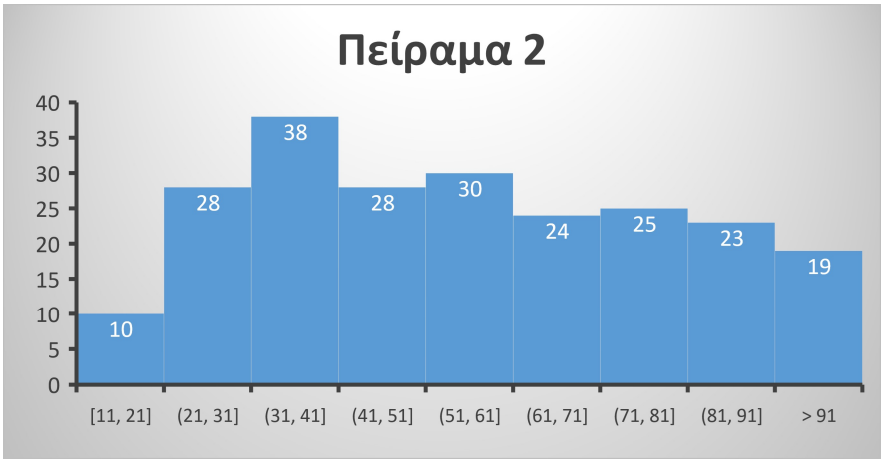
Μέσα στη λύση, όπως βλέπουμε, υπάρχουν και κινήσεις οι οποίες δεν επηρεάζουν τον πίνακα, πράγμα που δεν αποτελεί πρόβλημα καθώς σε όποια πλατφόρμα και να παίξει κάποιος το συγκεκριμένο παιχνίδι οι κινήσεις αυτές απλά παραλείπονται. Επίσης, υπάρχουν κινήσεις και μετά τη λύση τις οποίες όμως δε λαμβάνουμε υπόψη αφού αυτό που μας ενδιαφέρει είναι να φτάσουμε στη λύση.

Πειραματικά Αποτελέσματα

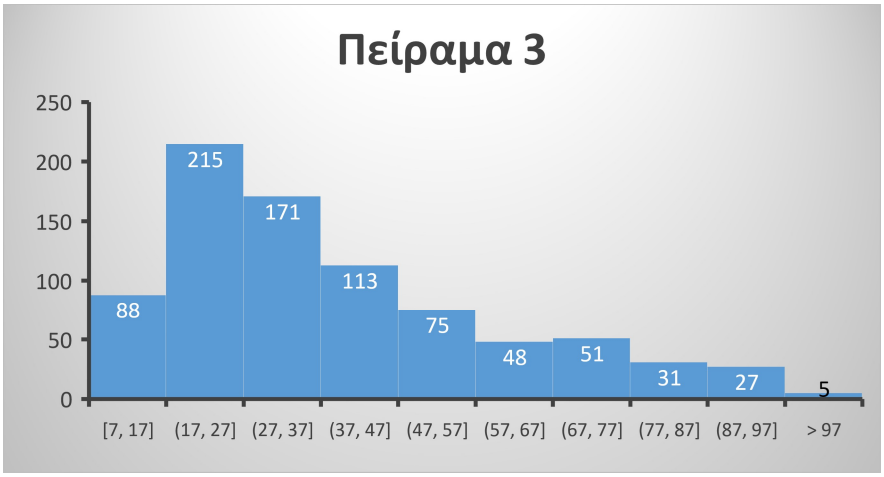
Στα ιστογράμματα που ακολουθούν φαίνεται για κάθε διαφορετικό αρχείο, μόνο για τις επιτυχημένες εκτελέσεις του αλγόριθμου, ο αριθμός γενεών που χρειάστηκε ο αλγόριθμος για να βρει την λύση.



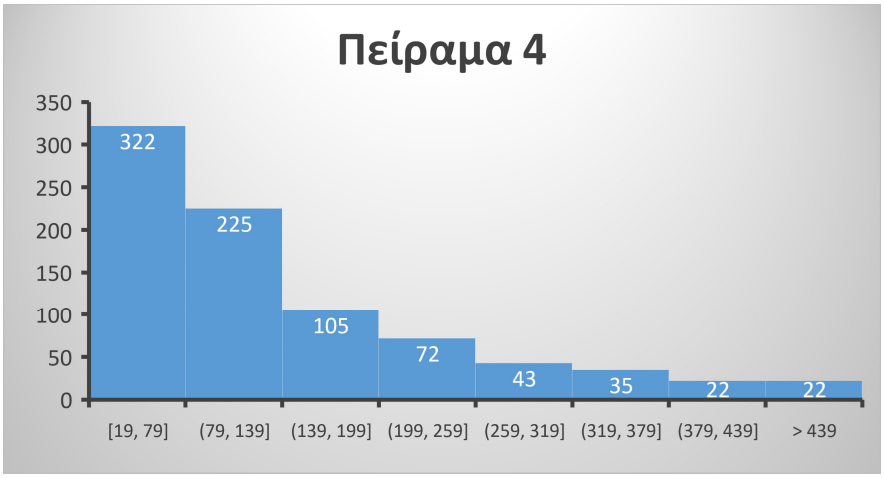
Εικόνα 15 Ιστόγραμμα του SameGame για το 1ο πείραμα



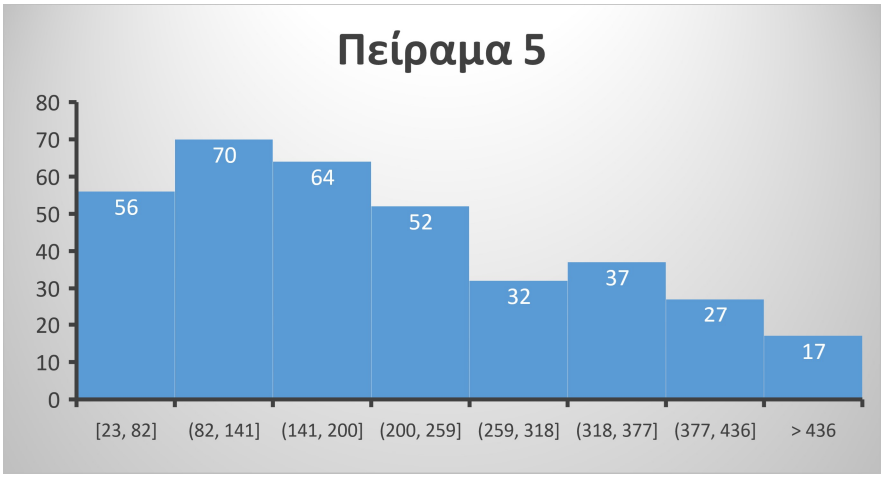
Εικόνα 16 Ιστόγραμμα του SameGame για το 2ο πείραμα



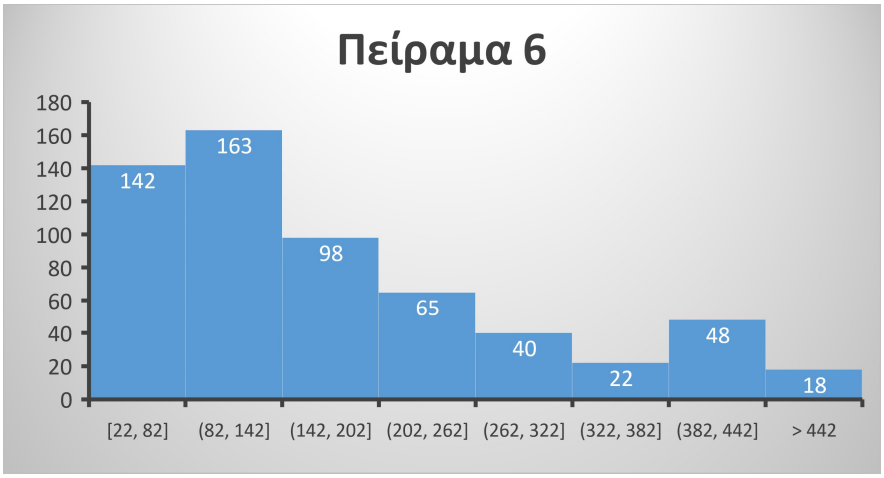
Εικόνα 17 Ιστόγραμμα του SameGame για το 3ο πείραμα



Εικόνα 18 Ιστόγραμμα του SameGame για το 4ο πείραμα



Εικόνα 19 Ιστόγραμμα του SameGame για το 5ο πείραμα



Εικόνα 20 Ιστόγραμμα του SameGame για το 6ο πείραμα

Στη συνέχεια χρησιμοποιήσαμε πάλι την κλάση σε Java που δημιουργήσαμε για την ανάλυση των αποτελεσμάτων του προηγούμενου παραδείγματος, προσαρμόζοντας την βέβαια στα δεδομένα του καινούριου, και τα αποτελέσματα που προέκυψαν φαίνονται στον παρακάτω πίνακα.

Πείραμα	***Ποσοστό Επιτυχίας***	***Διάμεσος***	***Μέσος***	***Τυπική Απόκλιση***
Εύκολο 1	90.4 %	30	34.03	17.92
Εύκολο 2	22.5 %	55	55.97	23.21
Εύκολο 3	82.4 %	33	39.38	21.40
Μεσαίο 1	84.6 %	100	139.79	109.80
Μεσαίο 2	35.5 %	185	208.86	121.75
Μεσαίο 3	59.6 %	140	177.98	119.33

Πίνακας 2 Ανάλυση αποτελεσμάτων του SameGame

Συμπέρασμα

Το πρώτο πράγμα που βλέπουμε από τον πίνακα είναι ότι ακόμα και μεταξύ όμοιων επίπεδων υπάρχει αρκετά μεγάλη απόκλιση στο ποσοστό επιτυχίας. Αυτό δείχνει ότι πέρα από την αύξηση του αριθμού των διαφορετικών χρωμάτων, στη δυσκολία επίλυσης, παίζει ρόλο και η διάταξη των αριθμών μέσα στον πίνακα. Μπορούμε να ισχυριστούμε ότι όσο περισσότερο είναι ομαδοποιημένοι οι αριθμοί, τόσο πιο εύκολο είναι και να λυθεί. Βέβαια, ο ισχυρισμός αυτός είναι διαισθητικός. Για την απόδειξή του θα χρειαζόταν να γίνει κάποιου τύπου ομαδοποίηση (clustering) των στοιχείων στον πίνακα ώστε να φανεί κατά πόσον ευσταθεί αυτός ο ισχυρισμός.

Όπως και στο προηγούμενο πρόβλημα, τα ιστογραμμάτα συχνότητας στην πλειονότητά τους είναι δεξιά ή αλλιώς θετικά ασύμμετρα (right skewed). Κατά συνέπεια φαίνεται ότι ο μεγαλύτερος όγκος των αποτελεσμάτων βρίσκεται σε μικρότερες τιμές του αριθμού των γενεών.

Τέλος οι σχετικά χαμηλές τιμές της διαμέσου για τα διαφορετικά πειράματα δείχνουν μία καλή συμπεριφορά της ΓΕ στο συγκεκριμένο πρόβλημα.

6. Γενικά συμπεράσματα

Μετά την ολοκλήρωση της παρούσας εργασίας και μελέτης πάνω στην ΓΕ μπορούμε να εξάγουμε κάποια γενικά συμπεράσματα.

Το GEVA, το περιβάλλον που χρησιμοποιήσαμε για την εφαρμογή της ΓΕ σε προβλήματα, είναι ένα πολύ απλό και εύκολο σε χρήση περιβάλλον που ταυτόχρονα παρέχει πολλές δυνατότητες. Μπορεί λοιπόν να αποτελέσει ένα εργαλείο μάθησης καθώς η ενασχόληση με αυτό δίνει στον χρήστη πολλές γνώσεις για την ΓΕ και τον τρόπο λειτουργίας της.

Παρόλα αυτά ένα σημαντικό μειονέκτημα του GEVA, που συναντήσαμε είναι ο περιορισμός του ως προς την γραμματική εισόδου, η οποία πρέπει να εκφραστεί υποχρεωτικά σε BNF (μορφή Backus-Naur). Αυτό συνεπάγεται ότι οι υποψήφιες λύσεις μπορούν να προκύπτουν μόνο από μία γραμματική χωρίς συμφραζόμενα, πράγμα που δεν μας επιτρέπει να εισάγουμε σε αυτή περιορισμούς που ενδεχομένως να απαιτούνται από το πρόβλημα. Επί πλέον το περιβάλλον δεν παρέχει ευκολία για χρήση ιδιαίτερων γραμματικών όπως οι κατηγορικές, οι δισδιάστατες ή άλλες γραμματικές.

Κάτι άλλο πολύ σημαντικό που πρέπει να αναφερθεί, είναι ο βαθμός δυσκολίας που έχει η διαδικασία επίλυσης ενός προβλήματος με ΓΕ και πιο συγκεκριμένα με τη χρήση του GEVA. Μπορεί κανείς να ισχυριστεί ότι εφόσον το GEVA παρέχει έτοιμο τον κώδικα για τη λειτουργία της ΓΕ και απαιτεί μόνο τον καθορισμό της γραμματικής, της αντικειμενικής συνάρτησης και των παραμέτρων, η επίλυση ενός προβλήματος με τη χρήση αυτού του περιβάλλοντος είναι εύκολη. Στην πραγματικότητα όμως αυτό δεν ισχύει καθώς η δυσκολία επίλυσης οποιουδήποτε προβλήματος βρίσκεται στα τρία αυτά πράγματα που απαιτεί για είσοδο το GEVA.

Κλείνοντας, το σημαντικότερο συμπέρασμα είναι ότι η ΓΕ δείχνει ότι αποτελεί ένα ισχυρό εργαλείο για τη λύση των προβλημάτων που επιλέξαμε. Πιστεύουμε, ότι αξίζει στη συνέχεια να μελετηθεί η χρήση της για την επίλυση παιχνιδιών γρίφου μεγαλύτερης διάστασης καθώς και παιχνιδιών συνδιαστικής βελτιστοποίησης ή γενικότερα προβλημάτων συνδυαστικής βελτιστοποίησης. Τέλος, είναι σημαντικό να μελετηθούν οι τύποι των γραμματικών, των μορφών αντικειμενικής συνάρτησης και των παραμέτρων που είναι οι καταλληλότερες για την επίλυση συγκεκριμένων κλάσεων προβλημάτων.

ΠΑΡΑΡΤΗΜΑ

1. Κλάση αντικειμενικής συνάρτησης των Πύργων του Ανόι

```
1.  package FitnessEvaluation.Hanoi;
2.
3.  import Exceptions.BadParameterException;
4.  import Individuals.Individual;
5.  import Individuals.Phenotype;
6.  import java.util.Properties;
7.
8.  import FitnessEvaluation.FitnessFunction;
9.  import Util.Constants;
10. import java.util.*;
11.
12. /**
13.  * Hanoi.java
14.  *
15.  * Created by Nikos Kornelakis on Jun 12, 2017, 13:54 PM
16.  *
17.  */
18.
19. public class Hanoi implements FitnessFunction {
20.
21.     private static final double MAX_FITNESS = 500.0;
22.     private static final int DISKS = 3;
23.     private static Stack<Integer>[] tower;
24.     private static double columnWeights[];
25.
26.
27.     /** Creates a new instance of Hanoi */
28.     public Hanoi()
29.     {
30.         columnWeights = new double[3];
31.         columnWeights[0] = 2;
32.         columnWeights[1] = 1;
33.         columnWeights[2] = 0;
34.     }
35.
36.     private void initialisation()
37.     {
38.         // Initialise the 3 towers
39.         tower = new Stack[3];
40.
41.         tower[0] = new Stack<Integer>();
42.         tower[1] = new Stack<Integer>();
43.         tower[2] = new Stack<Integer>();
44.
45.         for(int i=DISKS;i>0;i--)
46.             tower[0].push(i);
47.     }
48.
49.     public void setProperties(Properties p) {}
50.
51.     public double evaluateString(Phenotype p) {
52.         String pS = p.getStringNoSpace();
53.         double fitness = MAX_FITNESS;
54.
55.         initialisation();
56.
57.         String[] inputCommands = pS.split(";");
58.         int errorCount = 0;
59.         for (String command: inputCommands)
```

```
60.         {
61.             // position 0: from, position 1: to
62.             String[] fromTo = command.split(",");
63.             int from, to;
64.             try
65.             {
66.                 from = Integer.parseInt(fromTo[0]);
67.                 to = Integer.parseInt(fromTo[1]);
68.             }
69.             catch (Exception e)
70.             {
71.                 return fitness;
72.             }
73.
74.             if(hasDisks(from) && isSmallerThanTopTo(from, to))
75.                 moveDisk(from, to);
76.             else
77.                 errorCount++;
78.         }
79.
80.         fitness = stateEvaluation(tower);
81.
82.         //Increase fitness for errors at the evaluation
83.         fitness += errorCount;
84.
85.         return fitness;
86.     }
87.
88.      private boolean hasDisks(int col)
89.     {
90.         if(tower[col-1].size() == 0)
91.             return false;
92.
93.         return true;
94.     }
95.
96.     private boolean isSmallerThanTopTo(int source, int dest)
97.     {
98.         if(tower[dest-1].size() == 0)
99.             return true;
100.
101.                return (Integer.valueOf(tower[source-1].get(tower[source-
    1].size()-1)) < Integer.valueOf(tower[dest-1].get(tower[dest-1].size()-1)));
102.            }
103.
104.            private void moveDisk(int source, int dest)
105.            {
106.                int temp = tower[source-1].pop();
107.                tower[dest-1].push(temp);
108.            }
109.
110.            public double stateEvaluation(Stack<Integer>[] input)
111.            {
112.                double fit = 0;
113.                for(int i = DISKS; i > 0; i--)
114.                {
115.                    int d1, d2, d3;
116.                    try
117.                    {
118.                        d1 = Integer.valueOf(tower[0].get(i-1));
119.                        fit += d1 * columnWeights[0];
120.                    }
121.                    catch (Exception e){}
122.                    try
123.                    {
124.                        d2 = Integer.valueOf(tower[1].get(i-1));
```

```
125.                         fit += d2 * columnWeights[1];
126.                     }
127.                     catch(Exception e){}
128.                     try
129.                     {
130.                         d3 = Integer.valueOf(tower[2].get(i-1));
131.                         fit += d3 * columnWeights[2];
132.                     }
133.                     catch (Exception e){}
134.                 }
135.                 return fit;
136.             }
137.
138.             public void getFitness(Individual i) {
139.                 i.getFitness().setDouble(this.evaluateString(i.getPhenotype()));
140.             }
141.
142.             public boolean canCache()
143.             {
144.                 return true;
145.             }
146.
147.         }
```

2. Κλάση αντικειμενικής συνάρτησης του Mastermind

```
1.  package FitnessEvaluation.Mastermind;
2.
3.  import Exceptions.BadParameterException;
4.  import Individuals.Individual;
5.  import Individuals.Phenotype;
6.  import java.util.Properties;
7.
8.  import FitnessEvaluation.FitnessFunction;
9.  import Util.Constants;
10. import java.util.ArrayList;
11.
12. /**
13.  * Mastermind.java
14.  *
15.  * Created by Nikos Kornelakis on Aug 13, 2017, 12:08 PM
16.  *
17.  */
18.
19. public class Mastermind implements FitnessFunction {
20.
21.     private static String code;
22.     private static double guessPenalty = 1.0;
23.     private static double rightNumberBonus = 5.0;
24.     private static double rightPositionBonus = 10.0;
25.     private static int numberOfDigits = 4;
26.
27.     /** Creates a new instance of Mastermind */
28.     public Mastermind() {
29.         code = "1234";
30.     }
31.
32.     /**
33.      * Creates a new instance of Mastermind
34.      * @param s String to match
35.      */
36.     public Mastermind(String s) {
37.         Mastermind.code = s;
38.     }
39.
40.     public void setProperties(Properties p) {
41.         String key = Constants.MASTERMIND_CODE;
42.         String value;
43.         value = p.getProperty(key,"1234");
44.         Mastermind.code = value;
45.     }
46.
47.     public double evaluateString(Phenotype p) {
48.         String pS = p.getStringNoSpace();
49.         double totalFitness = 0;
50.         ArrayList<Character> codeList = new ArrayList<Character>();
51.         double fitness = 60;
52.         int guessCount;
53.         String[] guesses = pS.split(";");
54.         for (String guess: guesses)
55.         {
56.             fitness = 60.0;
57.             guessCount = 0;
58.             //check how many numbers are in right position
59.             for(int i=0;i<numberOfDigits;i++)
60.             {
61.                 if(guess.charAt(i) == code.charAt(i))
62.                     fitness -= rightPositionBonus;
63.             }
```

```
64.             //check how many numbers are in the solution
65.             codeList.clear();
66.             for(int i=0;i<numberOfDigits;i++)
67.                 codeList.add(code.charAt(i));
68.             for(int i=0;i<numberOfDigits;i++)
69.             {
70.                 for(int j=0;j<codeList.size();j++)
71.                 {
72.                     if(guess.charAt(i) == codeList.get(j))
73.                     {
74.                         codeList.remove(j);
75.                         fitness -= rightNumberBonus;
76.                         break;
77.                     }
78.                 }
79.             }
80.             guessCount++;
81.             if(fitness == 0.0)
82.                 break;
83.             if(guessCount < 10)
84.                 totalFitness++;
85.             else
86.                 totalFitness += fitness;
87.         }
88.         if(guesses.length <= 10)
89.             totalFitness += fitness;
90.
91.         return totalFitness;
92.     }
93.
94.     public void getFitness(Individual i) {
95.         i.getFitness().setDouble(this.evaluateString(i.getPhenotype()));
96.     }
97.
98.     public boolean canCache()
99.     {   return true;
100.         }
101.
102.        }
```

3. Batch file

```
@echo off
echo Please wait while genetic algorithm is running ...
echo Mastermind Problem with code 3182
FOR /L %%x IN (1,1,1000) DO (
cd C:\Users\Nikos\Desktop\GEVA-v2.0\bin
java -jar GEVA.jar -main_class Main.Run -
properties_file ../param/Parameters/Mastermind.properties -
code 3182>> C:\Users\Nikos\Desktop\MastermindData\code3182.txt
echo %%x time succeeded
)
echo Mastermind Problem with code 5337
FOR /L %%x IN (1,1,1000) DO (
cd C:\Users\Nikos\Desktop\GEVA-v2.0\bin
java -jar GEVA.jar -main_class Main.Run -
properties_file ../param/Parameters/Mastermind.properties -
code 5337>> C:\Users\Nikos\Desktop\MastermindData\code5337.txt
echo %%x time succeeded
)
echo Mastermind Problem with code 4441
FOR /L %%x IN (1,1,1000) DO (
cd C:\Users\Nikos\Desktop\GEVA-v2.0\bin
java -jar GEVA.jar -main_class Main.Run -
properties_file ../param/Parameters/Mastermind.properties -
code 4441>> C:\Users\Nikos\Desktop\MastermindData\code4441.txt
echo %%x time succeeded
)
echo Mastermind Problem with code 8171
FOR /L %%x IN (1,1,1000) DO (
cd C:\Users\Nikos\Desktop\GEVA-v2.0\bin
java -jar GEVA.jar -main_class Main.Run -
properties_file ../param/Parameters/Mastermind.properties -
code 8171>> C:\Users\Nikos\Desktop\MastermindData\code8171.txt
echo %%x time succeeded
)
echo Mastermind Problem with code 2617
FOR /L %%x IN (1,1,1000) DO (
cd C:\Users\Nikos\Desktop\GEVA-v2.0\bin
java -jar GEVA.jar -main_class Main.Run -
properties_file ../param/Parameters/Mastermind.properties -
code 2617>> C:\Users\Nikos\Desktop\MastermindData\code2617.txt
echo %%x time succeeded
)
pause
```

4. Java κλάση στατιστικής ανάλυσης

```
1.  import java.io.BufferedReader;
2.  import java.io.BufferedWriter;
3.  import java.io.FileReader;
4.  import java.io.FileWriter;
5.  import java.io.PrintWriter;
6.
7.  /**
8.  * Analysis.java
9.  *
10. * Created by Nikos Kornelakis on Aug 28, 2017, 15:47 PM
11. *
12. */
13.
14.
15. public class Analysis {
16.
17.     public static void main(String args[]){
18.
19.         final int executionsInFile = 1000;
20.         final int GENERATIONS = 100;
21.         final String CODE = "8171";
22.
23.         int timesSolved = 0;
24.         int generationsTable[] = new int [executionsInFile];
25.
26.         try
27.         {
28.             BufferedReader in = new BufferedReader(new FileReader("code"+CODE+".t
    xt"));
29.             String line;
30.
31.             //Iteration for every line
32.             while((line = in.readLine()) != null)
33.             {
34.                 line = line.replace(',','.');
35.                 String[] lineStats = line.trim().split("\\s+");
36.
37.                 //If it is not statistic line go to next iteration
38.                 if(lineStats.length != 10)
39.                     continue;
40.
41.                 int generation = -999;
42.                 double fitness = -999;
43.
44.                 try
45.                 {
46.                     generation = Integer.parseInt(lineStats[0]);
47.                     fitness = Double.parseDouble(lineStats[4]);
48.                 }
49.                 catch (Exception e) {}
50.
51.                 if(generation == GENERATIONS || fitness == 0.0){
52.
53.                     if(fitness == 0.0){
54.                         generationsTable[timesSolved] = generation;
55.                         timesSolved++;
56.
57.                     }
58.
59.                     if (generation == GENERATIONS && fitness != 0.0){
60.                         line = in.readLine();
61.                         String newLine = line.trim().replaceAll("\\s+","");
```

```
                        String resultLine[] = newLine.split(":");
                        String guesses[] = resultLine[resultLine.length-
1].split(";");
                        if(guesses.length <= 10){
                            for (String guess: guesses) {
                                if(guess.equals(CODE)){
                                    generationsTable[timesSolved] = generation;
                                    timesSolved++;
                                    break;
                                }
                            }
                        }
                    }
                }
            }

            in.close();
            System.out.println("Executions : "+executionsInFile+"  Solved : "+tim
esSolved);
        }
        catch (Exception e) {
            System.out.println(e.toString());
        }

        //calculating mean
        int meanSum = 0;
        for(int i=0; i<timesSolved; i++)
            meanSum += generationsTable[i];
        double mean = (double)meanSum / timesSolved;
        System.out.println("Mean : " + mean);

        //calculating standard deviation
        double standSum = 0;
        for(int i=0; i<timesSolved; i++)
            standSum += Math.pow((generationsTable[i] - mean), 2);

        double standDev = Math.sqrt(standSum / timesSolved);
        System.out.println("Standard Deviation : " + standDev);

        //store data to file before we sort them
                //storeFile(generationsTable, timesSolved, CODE);

                //sort table to find median
                for(int i=0; i<timesSolved; i++){
                    for(int j=timesSolved-1; j>i; j--){
                        if(generationsTable[j] < generationsTable[j-1]){
                            int temp = generationsTable[j];
                            generationsTable[j] = generationsTable[j-1];
                            generationsTable[j-1] = temp;
                        }
                    }
                }

                double median;
                if(timesSolved % 2 == 0){
                    median = (generationsTable[timesSolved / 2] + generationsTable
[(timesSolved / 2) - 1]) / 2;
                }
                else{
                    median = generationsTable[timesSolved / 2];
                }
                System.out.println("Median : " + median);

                //store the statistics to file
                storeStatisticsFile(executionsInFile, timesSolved, median, mean, s
tandDev, CODE);
```

```
124.
125.
126.            }
127.
128.            public static void storeFile(int[] input, int length, String code){
129.                try{
130.                    PrintWriter out = new PrintWriter(new BufferedWriter(new FileW
    riter("GenerationsFile"+code+".txt", true)));
131.                    for(int i=0; i<length; i++)
132.                        out.print(input[i]+" ");
133.                    out.println();
134.                    out.close();
135.                } catch (Exception e){
136.                    System.out.println(e.toString());
137.                    return;
138.                }
139.                System.out.println("Table stored in the file successfully");
140.            }
141.
142.            public static void storeStatisticsFile(int executions, int solved, dou
    ble median, double mean, double standDev, String code){
143.                try{
144.                    PrintWriter out = new PrintWriter("StatisticsFor"+code+".txt")
    ;
145.                    out.println("Total executions : " + executions);
146.                    out.println("Times solved : " + solved);
147.                    out.println("Median : " + median);
148.                    out.println("Mean : " + mean);
149.                    out.println("Standard Deviation : " + standDev);
150.                    out.println();
151.                    out.close();
152.                } catch (Exception e){
153.                    System.out.println(e.toString());
154.                    return;
155.                }
156.                System.out.println("Table stored in the file successfully");
157.            }
158.        }
```

5. Κλάση αντικειμενικής συνάρτησης του SameGame

```
1.  package FitnessEvaluation.SameGame;
2.
3.  import Exceptions.BadParameterException;
4.  import Individuals.Individual;
5.  import Individuals.Phenotype;
6.  import java.util.Properties;
7.
8.  import FitnessEvaluation.FitnessFunction;
9.  import Util.Constants;
10. import java.util.*;
11.
12. import java.io.BufferedWriter;
13. import java.io.File;
14. import java.io.FileNotFoundException;
15. import java.io.FileWriter;
16. import java.io.PrintWriter;
17. import java.util.ArrayList;
18. import java.util.NoSuchElementException;
19. import java.util.Random;
20.
21. /**
22.  * SameGame.java
23.  *
24.  * Created by Nikos Kornelakis on Aug 31, 2017, 14:10 PM
25.  *
26.  */
27.
28. public class SameGame implements FitnessFunction {
29.
30.     final int xArraySize = 9;
31.     final int yArraySize = 9;
32.     int array[][] = new int[xArraySize][yArraySize];
33.     private final double MAX_FITNESS = xArraySize * yArraySize;
34.     String file;
35.
36.
37.     /** Creates a new instance of SameGame */
38.     public SameGame()
39.     {
40.         file = "SameGame.txt";
41.     }
42.
43.     public void setProperties(Properties p) {
44.         String key = Constants.SAME_GAME_FILE;
45.         String value;
46.         value = p.getProperty(key,"SameGame.txt");
47.         file = value;
48.     }
49.
50.     private void initialiseArray()
51.     {
52.         //Read the random numbers for the array from the file
53.         try
54.         {
55.             Scanner scanner = new Scanner(new File(file));
56.             for(int i=0; i<xArraySize; i++) {
57.                 for(int j=0; j<yArraySize; j++)
58.                     array[i][j] = scanner.nextInt();
59.             }
60.             scanner.close();
61.         }
62.         catch (NoSuchElementException e){
```

```
63.             System.out.println("Error : Missing numbers -
    Array has size "+xArraySize+"x"+yArraySize+".");
64.             System.exit(0);
65.         }
66.         catch (FileNotFoundException e){
67.             System.out.println("Error : File could not found.");
68.             System.exit(0);
69.         }
70.         catch (Exception e)
71.         {
72.             System.out.println(e.toString());
73.             System.exit(0);
74.         }
75.     }
76.
77.     public double evaluateString(Phenotype p) {
78.         String pS = p.getStringNoSpace();
79.         double fitness = MAX_FITNESS;
80.
81.         initialiseArray();
82.
83.         //int indifferentMovesCount = 0;
84.         String[] inputCommands = pS.split(";");
85.         for (String coord: inputCommands)
86.         {
87.
88.         //here was while !gameOver()
89.
90.             int xCoord = 0, yCoord = 0;
91.             try
92.             {
93.                 xCoord = Integer.parseInt(coord.charAt(0)+"");
94.                 yCoord = Integer.parseInt(coord.charAt(1)+"");
95.             }
96.             catch(Exception e){
97.                 System.out.println("Error : Inccorect choice.");
98.                 System.exit(0);
99.             }
100.
101.                 //if its already empty or hasn't other same colors around the
    choice is skipped
102.                 if(array[xCoord][yCoord] == 0 || isSingleSquare(xCoord, yCoord
    ))
103.                 {
104.                     //indifferentMovesCount++;
105.                     continue;
106.                 }
107.
108.                 //delete the related squares
109.                 ArrayList<String> list = new ArrayList<String>();
110.                 list.add(xCoord+""+yCoord);
111.                 for(int i=0; i<list.size(); i++)
112.                 {
113.                     try
114.                     {
115.                         int x = Integer.parseInt(list.get(i).charAt(0)+"");
116.                         int y = Integer.parseInt(list.get(i).charAt(1)+"");
117.                         ArrayList<String> tempList = findNextSquares(x, y);
118.
119.                         for(int j=0; j<tempList.size(); j++)
120.                         {
121.                             if(!list.contains(tempList.get(j)))
122.                                 list.add(tempList.get(j));
123.                         }
124.                     }
125.                     catch(Exception e){}
```

```
126.                     }
127.                     //fitness += Math.pow((list.size() - 1), 2);
128.                     removeTheSquares(list);
129.
130.                     //move other squares that left down and left
131.                     moveSquaresDown();
132.                     moveSquaresLeft();
133.                 }
134.
135.                 int count = 0;
136.                 for(int i=0; i<xArraySize; i++)
137.                     for(int j=0; j<yArraySize; j++)
138.                         if(array[i][j] != 0)
139.                             count++;
140.
141.                 return count;
142.             }
143.
144.             private boolean gameOver()
145.             {
146.                 boolean flag = true;
147.
148.                 for(int i=0; i<xArraySize; i++)
149.                     for(int j=0; j<yArraySize; j++)
150.                         if(array[i][j] != 0) {
151.                             flag = false;
152.                             break;
153.                         }
154.                 if(flag)
155.                     return flag;
156.
157.                 for(int i=0; i<xArraySize; i++)
158.                 {
159.                     for(int j=0; j<yArraySize; j++)
160.                     {
161.                         if(array[i][j] == 0)
162.                             continue;
163.                         if(!isSingleSquare(i, j))
164.                             return false;
165.                     }
166.                 }
167.
168.                 return true;
169.             }
170.
171.             private boolean isSingleSquare(int x, int y)
172.             {
173.                 //check the left square
174.                 if(y != 0)
175.                     if(array[x][y] == array[x][y-1])
176.                         return false;
177.                 //check the right square
178.                 if(y != (yArraySize - 1))
179.                     if(array[x][y] == array[x][y+1])
180.                         return false;
181.                 //check the up square
182.                 if(x != 0)
183.                     if(array[x][y] == array[x-1][y])
184.                         return false;
185.                 //check the down square
186.                 if(x != (xArraySize - 1))
187.                     if(array[x][y] == array[x+1][y])
188.                         return false;
189.
190.                 return true;
191.             }
```

```
192.
193.            private ArrayList<String> findNextSquares(int x, int y)
194.            {
195.                ArrayList<String> ret = new ArrayList<String>();
196.                //check the left square
197.                if(y != 0)
198.                    if(array[x][y] == array[x][y-1])
199.                        ret.add(x+""+(y-1));
200.                //check the right square
201.                if(y != (yArraySize - 1))
202.                    if(array[x][y] == array[x][y+1])
203.                        ret.add(x+""+(y+1));
204.                //check the up square
205.                if(x != 0)
206.                    if(array[x][y] == array[x-1][y])
207.                        ret.add((x-1)+""+y);
208.                //check the down square
209.                if(x != (xArraySize - 1))
210.                    if(array[x][y] == array[x+1][y])
211.                        ret.add((x+1)+""+y);
212.
213.                return ret;
214.            }
215.
216.            private void removeTheSquares(ArrayList<String> input)
217.            {
218.                for(int i=0; i<input.size(); i++)
219.                {
220.                    try
221.                    {
222.                        int x = Integer.parseInt(input.get(i).charAt(0)+"");
223.                        int y = Integer.parseInt(input.get(i).charAt(1)+"");
224.
225.                        array[x][y] = 0;
226.                    }
227.                    catch(Exception e){}
228.                }
229.            }
230.
231.            private void moveSquaresDown()
232.            {
233.                for(int j=0; j<yArraySize; j++)
234.                {
235.                    int zeroCount = 0;
236.                    for(int i=xArraySize-1; i>=0; i--)
237.                    {
238.                        if(array[i][j] == 0)
239.                        {
240.                            zeroCount++;
241.                            continue;
242.                        }
243.                        if(zeroCount != 0)
244.                        {
245.                            array[i+zeroCount][j] = array[i][j];
246.                            array[i][j] = 0;
247.                        }
248.                    }
249.                }
250.            }
251.
252.            private void moveSquaresLeft()
253.            {
254.                int zeroCount = 0;
255.                for(int j=0; j<yArraySize; j++)
256.                {
257.                    if(array[xArraySize-1][j] == 0)
```

```
                    {
                        zeroCount++;
                        continue;
                    }
                    if(zeroCount != 0)
                    {
                        for(int i=0; i<xArraySize; i++)
                        {
                            array[i][j-zeroCount] = array[i][j];
                            array[i][j] = 0;
                        }
                    }
                }
            }

            public void getFitness(Individual i) {
                i.getFitness().setDouble(this.evaluateString(i.getPhenotype()));
            }

            public boolean canCache()
            {   return true;
            }

        }
```

6. Απλή υλοποίηση του παιχνιδιού SameGame σε Java

- Η κύρια κλάση.

```
1.  import java.io.File;
2.  import java.io.FileNotFoundException;
3.  import java.util.ArrayList;
4.  import java.util.NoSuchElementException;
5.  import java.util.Scanner;
6.
7.  /**
8.   * TheGame.java
9.   *
10.  *
11.  * Created by Nikos Kornelakis on Sep 3, 2017, 13:31 PM
12.  *
13.  */
14.
15. public class TheGame {
16.
17.     final int xArraySize = 9;
18.     final int yArraySize = 9;
19.     int array[][] = new int[xArraySize][yArraySize];
20.
21.     public TheGame() {}
22.
23.     public void execute()
24.     {
25.         Scanner sc = new Scanner(System.in);
26.
27.         //Read the random numbers for the array from the file
28.         try
29.         {
30.             Scanner scanner = new Scanner(new File("SameGame1.txt"));
31.             for(int i=0; i<xArraySize; i++) {
32.                 for(int j=0; j<yArraySize; j++)
33.                     array[i][j] = scanner.nextInt();
34.             }
35.             scanner.close();
36.         }
37.         catch (NoSuchElementException e){
38.             System.out.println("Error : Missing numbers -
    Array has size "+xArraySize+"x"+yArraySize+".");
39.             System.exit(0);
40.         }
41.         catch (FileNotFoundException e){
42.             System.out.println("Error : File could not found.");
43.             System.exit(0);
44.         }
45.         catch (Exception e)
46.         {
47.             System.out.println(e.toString());
48.             System.exit(0);
49.         }
50.
51.         while(!gameOver())
52.         {
53.             //Show the array
54.             System.out.println();
55.             for(int i=0; i<xArraySize; i++) {
56.                 System.out.print(i+" |  ");
57.                 for(int j=0; j<yArraySize; j++){
58.                     if(array[i][j] == 0)
59.                         System.out.print("  ");
```

```
                    else if(array[i][j] == 1)
                        System.out.print("x ");
                    else if(array[i][j] == 2)
                        System.out.print("o ");
                    else if(array[i][j] == 3)
                        System.out.print("w ");
                    else if(array[i][j] == 4)
                        System.out.print("v ");
                }
                System.out.println();
            }
            System.out.println("-----------------------");
            System.out.print("     ");
            for(int j=0; j<yArraySize; j++)
                System.out.print(j+" ");

            System.out.print("\n\nMove : ");

            //Read the user's choice
            String coord = sc.nextLine();
            int xCoord = 0, yCoord = 0;
            try
            {
                xCoord = Integer.parseInt(coord.charAt(0)+"");
                yCoord = Integer.parseInt(coord.charAt(1)+"");
            }
            catch(Exception e){
                System.out.println("Error : Inccorect choice.");
                System.exit(0);
            }

            //if its already empty or hasn't other same colors around the choice is skipped
            if(array[xCoord][yCoord] == 0 || isSingleSquare(xCoord, yCoord)){
                continue;
            }

            //delete the related squares
            ArrayList<String> list = new ArrayList<String>();
            list.add(xCoord+""+yCoord);
            for(int i=0; i<list.size(); i++)
                   {
                       try
                       {
                           int x = Integer.parseInt(list.get(i).charAt(0)+"");
                           int y = Integer.parseInt(list.get(i).charAt(1)+"");
                           ArrayList<String> tempList = findNextSquares(x, y);

                           for(int j=0; j<tempList.size(); j++)
                           {
                               if(!list.contains(tempList.get(j)))
                                   list.add(tempList.get(j));
                           }
                       }
                       catch(Exception e){}
                   }

                   removeTheSquares(list);

                   //move other squares that left down and left
                   moveSquaresDown();
                   moveSquaresLeft();
               }
               sc.close();

               //Show the array
```

```
125.                System.out.println();
126.                for(int i=0; i<xArraySize; i++) {
127.                    System.out.print(i+" |  ");
128.                    for(int j=0; j<yArraySize; j++){
129.                        if(array[i][j] == 0)
130.                            System.out.print("  ");
131.                        else if(array[i][j] == 1)
132.                            System.out.print("x ");
133.                        else if(array[i][j] == 2)
134.                            System.out.print("o ");
135.                        else if(array[i][j] == 3)
136.                            System.out.print("w ");
137.                        else if(array[i][j] == 4)
138.                            System.out.print("v ");
139.                    }
140.                    System.out.println();
141.                }
142.                System.out.println("-----------------------");
143.                System.out.print("     ");
144.                for(int j=0; j<yArraySize; j++)
145.                    System.out.print(j+" ");
146.
147.            }
148.
149.            private boolean gameOver()
150.            {
151.                boolean flag = true;
152.
153.                for(int i=0; i<xArraySize; i++)
154.                    for(int j=0; j<yArraySize; j++)
155.                        if(array[i][j] != 0) {
156.                            flag = false;
157.                            break;
158.                        }
159.                if(flag)
160.                    return flag;
161.
162.                for(int i=0; i<xArraySize; i++)
163.                {
164.                    for(int j=0; j<yArraySize; j++)
165.                    {
166.                        if(array[i][j] == 0)
167.                            continue;
168.                        if(!isSingleSquare(i, j))
169.                            return false;
170.                    }
171.                }
172.
173.                return true;
174.            }
175.
176.            private boolean isSingleSquare(int x, int y)
177.            {
178.                //check the left square
179.                if(y != 0)
180.                    if(array[x][y] == array[x][y-1])
181.                        return false;
182.                //check the right square
183.                if(y != (yArraySize - 1))
184.                    if(array[x][y] == array[x][y+1])
185.                        return false;
186.                //check the up square
187.                if(x != 0)
188.                    if(array[x][y] == array[x-1][y])
189.                        return false;
190.                //check the down square
```

```
191.                 if(x != (xArraySize - 1))
192.                     if(array[x][y] == array[x+1][y])
193.                         return false;
194.
195.                 return true;
196.             }
197.
198.             private ArrayList<String> findNextSquares(int x, int y)
199.             {
200.                 ArrayList<String> ret = new ArrayList<String>();
201.                 //check the left square
202.                 if(y != 0)
203.                     if(array[x][y] == array[x][y-1])
204.                         ret.add(x+""+(y-1));
205.                 //check the right square
206.                 if(y != (yArraySize - 1))
207.                     if(array[x][y] == array[x][y+1])
208.                         ret.add(x+""+(y+1));
209.                 //check the up square
210.                 if(x != 0)
211.                     if(array[x][y] == array[x-1][y])
212.                         ret.add((x-1)+""+y);
213.                 //check the down square
214.                 if(x != (xArraySize - 1))
215.                     if(array[x][y] == array[x+1][y])
216.                         ret.add((x+1)+""+y);
217.
218.                 return ret;
219.             }
220.
221.             private void removeTheSquares(ArrayList<String> input)
222.             {
223.                 for(int i=0; i<input.size(); i++)
224.                 {
225.                     try
226.                     {
227.                         int x = Integer.parseInt(input.get(i).charAt(0)+"");
228.                         int y = Integer.parseInt(input.get(i).charAt(1)+"");
229.
230.                         array[x][y] = 0;
231.                     }
232.                     catch(Exception e){}
233.                 }
234.             }
235.
236.             private void moveSquaresDown()
237.             {
238.                 for(int j=0; j<yArraySize; j++)
239.                 {
240.                     int zeroCount = 0;
241.                     for(int i=xArraySize-1; i>=0; i--)
242.                     {
243.                         if(array[i][j] == 0)
244.                         {
245.                             zeroCount++;
246.                             continue;
247.                         }
248.                         if(zeroCount != 0)
249.                         {
250.                             array[i+zeroCount][j] = array[i][j];
251.                             array[i][j] = 0;
252.                         }
253.                     }
254.                 }
255.             }
256.
```

```
257.            private void moveSquaresLeft()
258.            {
259.                int zeroCount = 0;
260.                for(int j=0; j<yArraySize; j++)
261.                {
262.                    if(array[xArraySize-1][j] == 0)
263.                    {
264.                        zeroCount++;
265.                        continue;
266.                    }
267.                    if(zeroCount != 0)
268.                    {
269.                        for(int i=0; i<xArraySize; i++)
270.                        {
271.                            array[i][j-zeroCount] = array[i][j];
272.                            array[i][j] = 0;
273.                        }
274.                    }
275.                }
276.            }
277.        }
```

- Η main που διαχειρίζεται την παραπάνω κλάση.

```
1.  public class Main {
2.
3.      public static void main(String args[]){
4.
5.          TheGame ob = new TheGame();
6.          ob.execute();
7.
8.      }
9.  }
```

Βιβλιογραφία

[1] E. J. Anderson and M. C. Ferris, «Genetic Algorithms for Combinatorial Optimization: The Assembly Line Balancing Problem,» 1991.

[2] S. Nguyen, M. Zhang and M. Johnston, «A Genetic Programming Based Hyper-heuristic Approach for Combinatorial Optimisation,» Dublin, 2011.

[3] X. Xu, H. Rong, M. Trovati, M. Liptrott and N. Bessis, «CS-PSO: chaotic particle swarm optimization algorithm,» 03 October 2016. [Ηλεκτρονικό].

[4] M. C. P. S. A. R. B. Jarboui, «Combinatorial particle swarm optimization (CPSO) for partitional clustering problem,» *Applied Mathematics and Computation,* αρ. 192, pp. 337-345, 2007.

[5] M. Eddaly, B. Jarboui and P. Siarry, «Combinatorial particle swarm optimization for solving blocking flowshop scheduling problem,» *Journal of Computational Design and Engineering,* αρ. 3, pp. 295-311, 2016.

[6] R. Funaki, H. Takano and J. Murata, «Re-Labeling Differential Evolution for Combinatorial Optimization and Interactive Evolutionary Computation,» *SICE Journal of Control, Measurement, and System Integration,* τόμ. 9, p. 18–025, January 2016.

[7] B. Gasbaoui and B. Allaoua, «Ant Colony Optimization Applied on Combinatorial Problem for Optimal Power Flow Solution,» *Leonardo Journal of Sciences,* αρ. 14, pp. 1-17, January-June 2009.

[8] J. Alguilar, «A General Ant Colony Model to solve Combinatorial Optimization Problems,» *Revista Colombiana de Computacion,* αρ. 2, pp. 7-18.

[9] J. Yang and Y. Zhuang, «An improved ant colony optimization algorithm for solving a complex combinatorial optimization problem,» *Applied Soft Computing,* αρ. 10, pp. 653-660, 2010.

[10] E. Önder, M. Özdemir and B. F. Yildirim, «COMBINATORIAL OPTIMIZATION USING ARTIFICIAL BEE COLONY ALGORITHM AND PARTICLE SWARM OPTIMIZATION SUPPORTED GENETIC ALGORITHM,» *Kafkas University Journal of,* τόμ. 4, αρ. 6, pp. 59-70, 2013.

[11] J. Odili, M. N. M. Kahar, A. Noraziah and S. F. Kamarulzaman, «A comparative evaluation of swarm intelligence techniques for solving combinatorial optimization problems,» *International Journal of Advanced,* pp. 1-11, May-June 2017.

[12] Β. Ζησιμόπουλος, *Συνδυαστική Βελτιστοποίηση Σημειώσεις,* 2007.

[13] «IEEE Computational Intelligence Society,» IEEE, [Ηλεκτρονικό]. Available: http://cis.ieee.org/field-of-interest.html.

[14] Βασίλειος Γ. Καμπουρλάζος και Γ. Α. Παπακώστας, Εισαγωγή στην ΥΠΟΛΟΓΙΣΤΙΚΗ ΝΟΗΜΟΣΥΝΗ, 2015.

[15] «Γενετικοί Αλγόριθμοι - Βικιπαίδεια,» 2 Ιανουάριος 2017. [Ηλεκτρονικό]. Available: https://el.wikipedia.org/wiki/Γενετικοί_Αλγόριθμοι.

[16] Δ. Ψούνης, «ΠΛΗ31 Ενότητα 4: Γενετικοί Αλγόριθμοι».

[17] A. E. Baumal, J. J. McPhee and P. H. Calamai, «Application of genetic algorithms to the design optimization of an active vehicle suspension system,» *Computer methods in applied mechanics and engineerlng,* αρ. 163, pp. 87-94, 1998.

[18] R. Alkhatib, G. N. Jazar and M. F. Golnaraghi, «Optimal design of passive linear suspension using genetic algorithm,» *Journal of Sound and Vibration,* αρ. 275, pp. 665-691, 2004.

[19] K. Fujita, N. Hirokawa, S. Akagi, S. Kitamura and H. Yokohata, *MULTI-OBJECTIVE OPTIMAL DESIGN OF AUTOMOTIVE ENGINE USING GENETIC ALGORITHM,* Atlanta, 1998.

[20] C.-C. Tsai, H.-C. Huang and C.-K. Chan, «Parallel Elite Genetic Algorithm and Its Application to Global Path Planning for Autonomous Robot Navigation,» *IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS,* αρ. 58, OCTOBER 2011.

[21] J. Tu and S. X. Yangt, *Genetic Algorithm Based Path Planning for a Mobile Robot,* Taipei, 2003.

[22] Y. Hu and S. X. Yang, *A Knowledge Based Genetic A1gorith:m for Path Planning of a Mobile Robot.,* New Orleans, 2004.

[23] D. W. Corne, M. J. Oates and G. D. Smith, Telecommunications Optimization: Heuristic and Adaptive Techniques, 2000.

[24] Rui Jiang and K. Y. Szeto, *Extraction of Investment Strategies based on Moving Averages: A Genetic Algorithm Approach,* Hong Kong, 2003.

[25] Sam Mahfoud and G. Mani, «Financial forecasting using genetic algorithms,» *Applied Artificial Intelligence,* αρ. 10, pp. 543-566, 26 November 1996.

[26] R. J. Bauer, Genetic Algorithms and Investment Strategies by Richard, Wiley (1708), 1994.

[27] Sushil J. Louis and C. Miles, «Playing to Learn: Case-Injected Genetic Algorithms for Learning to Play Computer Games,» *IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION,* αρ. 9, DECEMBER 2005.

[28] Chris Miles S. J. Louis, N. Cole and J. McDonnell , *Learning to Play Like a Human: Case Injected Genetic Algorithms for Strategic Computer Gaming,* Reno, 2004.

[29] Nicholas Cole, S. J. Louis and C. Miles, *Using a Genetic Algorithm to Tune First-*

Person Shooter Bots, Reno, 2004.

[30] Σ. ΛΥΚΟΘΑΝΑΣΗΣ και Ε. Γεωργόπουλος, *Γενετικός Προγραμματισμός (ΓΠ),* Πάρτα, 2012.

[31] Ian Dempsey, M. O'Neill and A. Brabazon, Foundations in Grammatical Evolution for Dynamic Environments, Warsaw: Springer-Verlag Berlin Heidelberg, 2009.

[32] M. O'Neill, E. Hemberg, C. Gilligan, E. Bartley, J. McDermott and A. Barbazon, *GEVA - Grammatical Evolution in Java (v2.0),* Dublin, 2011.