ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ ΚΕΝΤΡΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΤΟΜΕΑΣ ΥΠΟΛΟΓΙΣΤΙΚΩΝ ΤΕΧΝΙΚΩΝ & ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΑΝΑΠΤΥΞΗ ΑΛΓΟΡΙΘΜΩΝ ΑΡΜΟΝΙΑΣ ΚΑΙ ΕΦΑΡΜΟΓΗ ΤΟΥΣ ΣΕ ΠΡΟΒΛΗΜΑΤΑ ΒΕΛΤΙΣΤΟΠΟΙΗΣΗΣ

Γιαννακούλης Αντώνιος

Επιβλέπων:
Δρ. Καζαρλής Σπυρίδων
Καθηγητής

ΣΕΡΡΕΣ 2017

Περίληψη

Η βελτιστοποίηση είναι μια πολύ σημαντική έννοια και διαδικασία που επηρεάζει σχεδόν κάθε κλάδο της επιστήμης. Για μερικά προβλήματα βελτιστοποίησης έχουν αναπτυχθεί μέθοδοι που μπορούν να τα επιλύσουν αποδοτικά. Ωστόσο, παρά την εξέλιξη των υπολογιστών, τα περισσότερα προβλήματα βελτιστοποίησης δεν μπορούν να επιλυθούν γρήγορα και με ακρίβεια. Η πολυπλοκότητά τους είναι τόσο μεγάλη που είναι αδύνατο να εντοπιστεί η βέλτιστη λύση γρήγορα. Στα προβλήματα μεγάλης πολυπλοκότητας η βέλτιστη λύση προσεγγίζεται στοχαστικά με χρήση ευριστικών μεθόδων. Οι περισσότεροι από τους ευριστικούς αλγορίθμους αντλούν έμπνευση από διαδικασίες και έννοιες της πραγματικής ζωής. Εξέλιξη, συμπεριφορά σμηνών, μεταλλουργία και μουσική είναι μόνο μερικές από τις έννοιες που έχουν δώσει έμπνευση στους πιο γνωστούς και επιτυχημένους ευριστικούς αλγορίθμους. Ένας τέτοιος αλγόριθμος είναι και η Αναζήτησης Αρμονίας. Η Αναζήτηση Αρμονίας αντλεί έμπνευση από την μουσική και συγκεκριμένα από τον τρόπο που αναζητείται η τέλεια αρμονία μέσα σε ένα μουσικό σχήμα. Η Αναζήτηση Αρμονίας αποτελείται από πολλούς επιμέρους μηχανισμούς που κατευθύνουν και προσανατολίζουν τον αλγόριθμο στην εύρεση του ολικού βέλτιστου μιας συνάρτησης. Για σχεδόν κάθε μηχανισμό του αλγορίθμου έχουν προταθεί κατά καιρούς διάφορες παραλλαγές.

Στην παρούσα εργασία, η απόδοση του αλγορίθμου Αναζήτησης Αρμονίας εκτιμάται από πρόγραμμα που δημιουργήθηκε με σκοπό την εφαρμογή του αλγορίθμου σε προβλήματα δοκιμών. Στο πρόγραμμα, δίνεται η δυνατότητα στον χρήστη να πειραματιστεί και να παρατηρήσει τα αποτελέσματα του αλγορίθμου για κάθε συνδυασμό παραμέτρων σε οποιαδήποτε μαθηματική συνάρτηση.

Abstract

Optimization is an essential notion and procedure that affects almost every area of interest in science. For some optimization problems there are methods which can solve these problems very quick. However, despite the fact that computers are becoming more and more powerful, most optimization problems cannot be solved efficiently. Their complexity is so enormous that it's impossible to find the optimum solution with precision in reasonable time. For that kind of complexity, the only viable approach is to find a near optimum solution using heuristic methods. Most popular and successful heuristic methods are inspired by nature or by real life procedures such as evolution, swarm intelligence, metallurgy and music. One such method is the Harmony Search. Harmony Search draws inspiration from the musical process of searching for a perfect state of harmony. Harmony Search consists of many mechanisms that can guide the search process to find the global optima. For almost every mechanism there are many variations proposed.

In this project, the performance of Harmony Search was evaluated after the development of an application which implements and tests the algorithm's performance on benchmark problems. The application gives the opportunity to the users to experiment and observe the results of the algorithm in any set of parameters in any mathematical function.

ΠΕΡΙΕΧΟΜΕΝΑ

Κεφάλαιο IV

Κεφάλαιο V

Κεφάλαιο VI

Κεφάλαιο VII

Κεφάλαιο VIII

Ι. ΒΕΛΤΙΣΤΟΠΟΙΗΣΗ

Επισκόπηση

Βελτιστοποίηση είναι μια διαδικασία, μέθοδος ή πράξη μέσω της οποίας βελτιώνεται όσο το δυνατόν περισσότερο ή τελειοποιείται ένα σύστημα, ένα αντικείμενο, μια διαδικασία, μια ιδέα.

Στα Μαθηματικά, η έννοια της βελτιστοποίησης, είναι η εύρεση των καλύτερων τιμών που δύναται να πάρει ως είσοδο μια συνάρτηση προκειμένου να παράξει το καλύτερο δυνατό αποτέλεσμα. Το καλύτερο αποτέλεσμα μιας συνάρτησης είναι η μέγιστη ή η ελάχιστη τιμή της. Το αν αναζητείται η μέγιστη ή η ελάχιστη τιμή εξαρτάται από το πρόβλημα που πρόκειται να αντιμετωπιστεί.

Φυσικά, η βελτιστοποίηση ως έννοια δεν συναντάται μόνο στα μαθηματικά. Όλοι οι άνθρωποι κάνουν καθημερινά υπολογισμούς βελτιστοποίησης. Ο υπολογισμός της βέλτιστης(συντομότερης) διαδρομής για την μετάβαση σε κάποιον προορισμό ή η κατανομή των πόρων για την επίτευξη ενός στόχου είναι διαδικασίες βελτιστοποίησης.

Ιστορία

Η εξερεύνηση προβλημάτων βελτιστοποίησης και η επιστημονική μελέτη τους ξεκίνησε από πολύ παλιά. Ήδη από την αρχαιότητα πολλοί ερευνητές ασχολήθηκαν με προβλήματα βελτιστοποίησης.

Ο **Ευκλείδης ο Αλεξανδρεύς(325 π.Χ – 265π.Χ)** στο τρίτο βιβλίο της μαθηματικής πραγματείας "Στοιχεία" διατυπώνει την απόδειξη για την μέγιστη και την ελάχιστη ευθεία που μπορεί να υπάρξει από ένα σημείο προς την περίμετρο ενός κύκλου. Στο έκτο βιβλίο της ίδιας πραγματείας παρουσιάζεται μια ακόμη απόδειξη σύμφωνα με την οποία το τετράγωνο έχει το μέγιστο εμβαδόν μεταξύ όλων των ορθογωνίων σχημάτων όταν είναι γνωστή η περίμετρος. Λίγα χρόνια αργότερα, ο **Αρχιμήδης ο Συρακούσιος(286 π.Χ. – 212 π.Χ.)** τελειοποίησε μια τεχνική ολοκλήρωσης που επέτρεπε τον υπολογισμό του εμβαδού και του όγκου πολλών σωμάτων χρησιμοποιώντας την μέθοδο της εξάντλησης. Με αυτή την μέθοδο ήταν δυνατό να υπολογιστούν καμπυλωτές περιοχές προσεγγιστικά αθροίζοντας συνεχόμενα εμβαδά πολυγώνων. Ο **Ήρων ο Αλεξανδρεύς(10 μ.Χ. - 75 μ.Χ.)** έδειξε ότι το φως ταξιδεύει από ένα σημείο σε ένα άλλο μέσω της ελάχιστης δυνατής διαδρομής. Ο **Πάππος ο Αλεξανδρεύς(290 μ.Χ – 350 μ.Χ.)** ήταν ο τελευταίος μεγάλος Έλληνας γεωμέτρης.

Απέδειξε(ελλιπώς) την γνωστή σήμερα *Εικασία της Κηρήθρας(Honeycomb Conjecture)*. Η απόδειξη ολοκληρώθηκε το 1999 από τον **Thomas Hales**. Αποδείχθηκε ότι τα κανονικά εξάγωνα είναι ο βέλτιστος τρόπος για να καταμεριστεί ένας χώρος σε ίσα τμήματα με την ελάχιστη δομική στήριξη. Στη πραγματικότητα, οι μέλισσες χρησιμοποιούν κηρήθρες για να αποθηκεύσουν μέλι. Οι κηρήθρες αυτές σχηματίζονται μέσα από πολλά εξάγωνα. Έτσι, οι μέλισσες μπορούν να αξιοποιούν πλήρως το διαθέσιμο χώρο τους, να παράγουν μια ελαφριά αλλά ανθεκτική κηρήθρα με τη λιγότερη δυνατή ποσότητα κεριού και να αποθηκεύουν τη μέγιστη ποσότητα μελιού στο δεδομένο χώρο.

Για πολλά χρόνια από το τέλος της αρχαιότητας και μέχρι το μέσο του μεσαίωνα υπάρχει ένα μεγάλο κενό στην ιστορία της βελτιστοποίησης(όπως άλλωστε και στις περισσότερες επιστήμες). Αρκετούς αιώνες αργότερα, ο **Ibn Sahl(940 – 1000)** υπολόγισε το βέλτιστο σχήμα που πρέπει να έχουν οι φακοί και τα κάτοπτρα. Πιθανόν να ήταν και η πρώτη εφαρμογή της βελτιστοποίησης σε πρόβλημα της μηχανικής. Ο **Al-Karaji(953 – 1029)** χρησιμοποίησε μια, όχι ολοκληρωμένη, απόδειξη μέσω επαγωγής για να υπολογίσει το άθροισμα των ολοκληρωμένων κύβων και να λύσει το διωνυμικό πρόβλημα. Μερικούς αιώνες αργότερα, Ο **Pierre de Fermat(1601 – 1665),** αφού πρώτα είχε ασχοληθεί με την αναλυτική Γεωμετρία, κατάφερε να βρει μια φόρμουλα που θα βρίσκει την παράγωγο και θα υπολογίζει το ακρότατο(βέλτιστο) σημείο όταν η κλίση είναι μηδέν. Αυτός ίσως ήταν και ο θεμέλιος λίθος των εφαρμοσμένων μαθηματικών και της χρήσης τους σε προβλήματα βελτιστοποίησης. Ο **Isaac Newton(1643 – 1727),** ένας από τους μεγαλύτερους επιστήμονες όλων των εποχών, υπολόγισε το βέλτιστο σχήμα που πρέπει να έχει μια σφαίρα. Ένα άλλο πρόβλημα βελτιστοποίησης που επίλυσε ήταν το πρόβλημα του *βράχιστου χρόνου(Brachistochrone)*. Με λίγα λόγια, κατάφερε να υπολογίσει την καμπύλη που βελτιστοποιεί την ταχύτητα πτώσης ενός σώματος - σημείου μεταξύ δύο σημείων, όταν η μόνη δύναμη που ασκείται είναι η βαρύτητα. Την ίδια εποχή, ο **Jacob Bernoulli(1654 – 1705)** έλυσε κάποια προβλήματα βελτιστοποίησης όπως η εύρεση της καμπύλης που πρέπει σχηματίζει μια ιδανική αλυσίδα υπό το ίδιο της το βάρος όταν στηρίζεται μόνο στα άκρα της(*Catenary*).

Για αρκετές ακόμα δεκαετίες και μέχρι τα μέσα του 20ού αιώνα η επίλυση προβλημάτων βελτιστοποίησης προσεγγιζόταν κυρίως μέσω της Μαθηματικής Ανάλυσης. Η έλευση των Υπολογιστών όμως άλλαξε τα πάντα. Οι μηχανές μπορούσαν να αντιμετωπίσουν προβλήματα Βελτιστοποίησης πολύ αποτελεσματικότερα από τον άνθρωπο. Η Ανάλυση έδωσε τη θέση της στους Αλγορίθμους. Ο Σοβιετικός επιστήμονας **Leonid Kantorovich(1912 – 1986)** εισήγαγε έναν αλγόριθμο Γραμμικού Προγραμματισμού για την βέλτιστη κατανομή πόρων. Το 1947, ο **George Dantzig(1914 – 2005)** δημοσίευσε την μέθοδο Simplex που χρησιμοποιείται ακόμα και σήμερα(ελαφρώς τροποποιημένος) σε εφαρμογές γραμμικής βελτιστοποίησης. Ο γραμμικός προγραμματισμός είναι μια μέθοδος για να επιτευχθεί το βέλτιστο αποτέλεσμα σε ένα μαθηματικό μοντέλο, όταν οι περιορισμοί εκφράζονται μέσα από γραμμικές εξισώσεις και η αντικειμενική συνάρτηση είναι γραμμική. Αποτέλεσε αντικείμενο εκτενούς έρευνας κατά τη διάρκεια του Β' Παγκοσμίου Πολέμου. Το 1939, ο **William Karush(1917 – 1997)** ήταν ο πρώτος που ανέπτυξε μεθόδους Μη Γραμμικού Προγραμματισμού. Ο Μη Γραμμικός

Προγραμματισμός έχει ιδιαίτερη σημασία στη Θεωρία Ελέγχου. Το 1952, ο **Richard Bellman(1920 – 1984)** ήταν ο πρώτος που δημοσίευσε έρευνες πάνω στον Δυναμικό Προγραμματισμό. Βασικό χαρακτηριστικό του Δυναμικού Προγραμματισμού είναι η βέλτιστη επίλυση πολύπλοκων προβλημάτων διαιρώντας τα σε μικρότερα και απλούστερα προβλήματα. Μέχρι αυτό το σημείο όλες οι μέθοδοι βελτιστοποίησης ήταν ντετερμινιστικές. Το 1952, δημοσιεύτηκε ο πρώτος αλγόριθμος στοχαστικής προσέγγισης της βέλτιστης λύσης από τους **Jacob Wolfowitz(1910 - 1981)** και **Jack Kiefer(1924 – 1981)**. Το 1953, ο Αμερικανός φυσικός **Nicholas Metropolis(1915 – 1999)** παρουσίασε σε διατριβή του μια τεχνική που θα αποτελούσε τον προπομπό του πιθανολογικού μεταευρετικού αλγορίθμου, *Simulated Annealing(1983)*. Τις δεκαετίες του ‘50 και του ‘60 γεννήθηκε ο τομέας της Τεχνητής Νοημοσύνης. Μαζί με την Τεχνητή Νοημοσύνη, δημιουργήθηκε μια μεγάλη οικογένεια μεταευρετικών αλγορίθμων, η Εξελικτική Υπολογιστική. Οι εξελικτικοί αλγόριθμοι χρησιμοποιούνται ακόμα και σήμερα για την επίλυση NP-Hard προβλημάτων.

Χαρακτηριστικά

➢ Περιορισμοί

Κάποια προβλήματα βελτιστοποίησης έχουν περιορισμούς και κάποια δεν έχουν. Όταν υπάρχουν περιορισμοί τότε χωρίζονται σε **μαλακούς περιορισμούς** και **σκληρούς περιορισμούς**. Σκληροί είναι οι περιορισμοί που είναι αναγκαίο να τηρηθούν. Μαλακοί είναι οι περιορισμοί που είναι καλό να τηρηθούν αλλά όχι αναγκαίο. Έστω ότι πρέπει να δημιουργηθεί ένα πρόγραμμα για ένα σχολείο. Σκληρός περιορισμός είναι να μην ανατεθεί η ίδια αίθουσα για 2 διαφορετικές διαλέξεις την ίδια μέρα και ώρα. Μαλακός περιορισμός θα μπορούσε να είναι η διεξαγωγή διαλέξεων σε “βολικές” ώρες εάν αυτό είναι δυνατό.

➢ Μεταβλητές Απόφασης

Είναι οι παράμετροι που δέχεται η αντικειμενική συνάρτηση. Η επιλογή των παραμέτρων είναι που καθορίζει πόσο καλή ή κακή είναι η λύση του προβλήματος. Κάποιες φορές οι **μεταβλητές απόφασης(decision variables)** έχουν προκαθορισμένο εύρος τιμών. Όταν συμβαίνει αυτό λέγονται **ντετερμινιστικές(deterministic)**. Αντίθετα, όταν μπορούν να λάβουν οποιαδήποτε τιμή ονομάζονται **στοχαστικές(stochastic)**. Για παράδειγμα αν θέλουμε να υπολογίσουμε την βέλτιστη κλίση/γωνία που μπορεί να έχει μια αεροτομή, ώστε να προσφέρει αεροδυναμικά οφέλη σε κάποιο όχημα, δεν έχει νόημα να εξεταστούν κάποιες πολύ μεγάλες ή πολύ

μικρές τιμές οπότε αποκλείονται εξ΄ αρχής από το πεδίο ορισμού. Επιπλέον, οι μεταβλητές απόφασης μπορούν να διακριθούν και από το είδος των τιμών που παίρνουν. Σε κάποια προβλήματα δέχονται μόνο ακέραιες τιμές ενώ σε άλλα μόνο πραγματικές. Αν οι επιτρεπόμενες τιμές είναι μόνο ακέραιες τότε πρόβλημα **Ακέραιου Προγραμματισμού(Integer Programming)**. Διαφορετικά, αν οι τιμές είναι πραγματικές τότε είναι πρόβλημα **Πραγματικών Τιμών(Real Values)**.

- **Δομή του προβλήματος**

Ανάλογα με τη δομή του εκάστοτε προβλήματος, τα προβλήματα βελτιστοποίησης μπορούν να διακριθούν ως προβλήματα **βέλτιστου(Optimal Control)** και **μη-βέλτιστου ελέγχου**. Σ' ένα πρόβλημα βέλτιστου ελέγχου ο αλγόριθμος που χρησιμοποιείται αποτελείται από πολλές γενιές. Κάθε γενιά εξελίσσεται από την προηγούμενη. Συνήθως περιγράφεται από δύο τύπους μεταβλητών, τις μεταβλητές ελέγχου και τις μεταβλητές κατάστασης. Οι μεταβλητές ελέγχου προσδιορίζουν το σύστημα και ελέγχουν την εξέλιξη του συστήματος από τη μια γενιά στην επόμενη. Οι μεταβλητές κατάστασης περιγράφουν τη συμπεριφορά ή την κατάσταση του συστήματος σε κάθε γενιά. Σκοπός του προβλήματος είναι να βρεθούν οι μεταβλητές ελέγχου που βελτιστοποιούν την αντικειμενική συνάρτηση. Παράδειγμα προβλήματος βέλτιστου ελέγχου αποτελεί η ελαχιστοποίηση του χρόνου διαδρομής ενός λεωφορείου που πραγματοποιεί κάποιο δρομολόγιο. Πώς πρέπει να χειρίζεται ο οδηγός την ταχύτητα και την επιτάχυνση του τρένου; Σε αυτό το πρόβλημα μεταβλητή ελέγχου είναι η επιτάχυνση και πέδηση που ασκεί ο οδηγός στο λεωφορείο, ενώ μεταβλητή κατάστασης είναι η ταχύτητα του λεωφορείου.

- **Αντικειμενική συνάρτηση**

Είναι η συνάρτηση που πρόκειται να βελτιστοποιηθεί και που περιγράφει πλήρως, με μαθηματικό τρόπο, το εκάστοτε πρόβλημα βελτιστοποίησης. Οι αντικειμενικές συναρτήσεις μπορούν διακριθούν σε **διακριτές** και **συνεχείς** ανάλογα με την τιμή που παίρνουν. Όταν η συνάρτηση είναι διακριτή τότε λαμβάνει ένα μόνο σύνολο τιμών. Αντίθετα όταν είναι συνεχής μπορεί να δώσει οποιαδήποτε τιμή. Όταν το πρόβλημα που αντιμετωπίζεται έχει μία μόνο αντικειμενική συνάρτηση τότε ονομάζεται **μονοκριτηριακό(single objective)**. Μερικά προβλήματα όμως απαιτούν παραπάνω από μία αντικειμενικές συναρτήσεις. Αυτά τα προβλήματα ονομάζονται **πολυκριτηριακά(multi objective)**. Έστω ότι πρέπει να κατανεμηθούν πόροι ώστε η παραγωγή μιας εταιρείας να έχει το βέλτιστο κέρδος. Σ' ένα τέτοιο πρόβλημα θα πρέπει να συνυπολογιστούν διαφορετικοί παράγοντες όπως η ποιότητα του προϊόντος, το κόστος παραγωγής και η ημερομηνία διάθεσης του προϊόντος. Προφανώς θα πρέπει να χρησιμοποιηθούν πολλές αντικειμενικές συναρτήσεις για να ληφθούν

υπόψη όλοι οι παράγοντες και να υπάρξει μια ισορροπημένη παραγωγή. Πολλές φορές όταν επιλύονται πολυκριτηριακά προβλήματα οι διαφορετικές αντικειμενικές συναρτήσεις είναι αλληλοσυγκρουόμενες. Για παράδειγμα όσο μειώνεται το κόστος παραγωγής τόσο χειρότερη θα είναι η ποιότητα του τελικού προϊόντος.

ΙΙ. ΥΠΟΛΟΓΙΣΤΙΚΗ ΠΟΛΥΠΛΟΚΟΤΗΤΑ

Ως **πολυπλοκότητα(complexity)** ορίζεται το κόστος χρήσης ενός αλγορίθμου από έναν υπολογιστή. Το κόστος ενός αλγορίθμου αναλύεται και ορίζεται μέσα από τις έννοιες της χρονικής και χωρικής πολυπλοκότητας. Χ**ρονική πολυπλοκότητα(Time Complexity)** είναι η μεταβολή του υπολογιστικού χρόνου ενός αλγορίθμου όταν μεταβάλλονται τα δεδομένα ενός προβλήματος. Αντίστοιχα, **Χωρική Πολυπλοκότητα(Space Complexity)** είναι η μεταβολή του χώρου(μνήμη που καταλαμβάνει ένα πρόγραμμα) όταν μεταβάλλονται τα δεδομένα ενός προβλήματος. Η έννοια της πολυπλοκότητας ενός αλγορίθμου αντιπροσωπεύει πλήρως το κόστος χρήσης του για την επίλυση ενός προβλήματος.

Με τον όρο **Υπολογιστική Πολυπλοκότητα(Computational Complexity)** ονομάζεται ο κλάδος της Πληροφορικής που ασχολείται με την ταξινόμηση υπολογιστικών προβλημάτων βάσει της πολυπλοκότητας που παρουσιάζουν. Μια κλάση πολυπλοκότητας αποτελείται από προβλήματα που έχουν όμοιες απαιτήσεις σε πόρους προκειμένου να επιλυθούν. Όσο περισσότερο εξερευνάται η φύση του υπολογισμού τόσο περισσότερες κλάσεις ανακαλύπτονται.

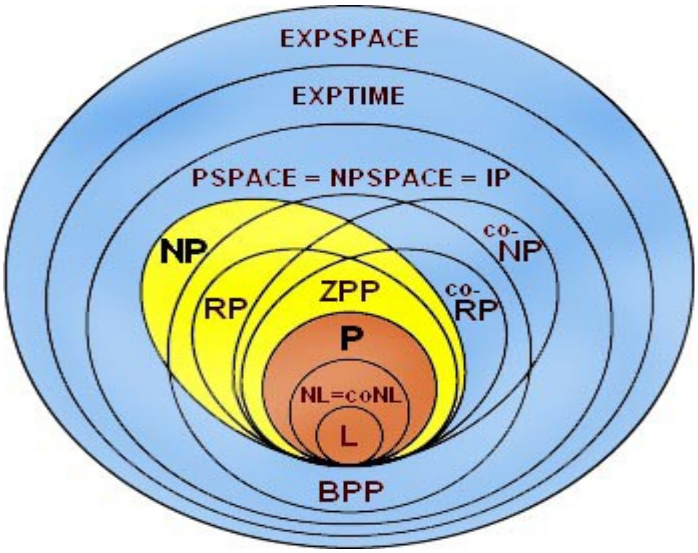
- **P(Deterministic Polynomial Time):** Τα προβλήματα που ανήκουν σε αυτή την κλάση μπορούν να υπολογιστούν σε πολυωνυμικό χρόνο. Αυτό σημαίνει ότι μπορούν να επιλυθούν σε ένα εύλογο χρονικό διάστημα από έναν αλγόριθμο. Μερικά παραδείγματα τέτοιων προβλημάτων είναι η εύρεση πρώτων αριθμών, η ταξινόμηση κ.λπ..

- **NP(Non-Deterministic Polynomial Time):** Σε αυτή την κλάση περιλαμβάνονται προβλήματα που αν δωθεί η λύση τους, τότε μπορεί να επαληθευτεί σε πολυωνυμικό χρόνο αν είναι σωστή ή όχι. Για την επίλυση προβλημάτων NP δεν υπάρχει γνωστός αλγόριθμος που να μπορεί να τα επιλύσει με μικρό κόστος. Ωστόσο, δεν έχει αποδειχθεί ότι είναι αδύνατο να υπάρξει ένας τέτοιος αλγόριθμος. Μερικά NP προβλήματα είναι το Πρόβλημα του Περιοδεύοντα Πωλητή, το Πρόβλημα του Σακιδίου κ.α..

- **PSPACE(Polynomial Space):** Είναι τα προβλήματα που μπορούν να επιλυθούν με πολυωνυμική μεταβολή του χώρου.

- **EXPTIME(Deterministic Exponential Time):** Είναι τα προβλήματα που είναι επιλύσιμα σε εκθετικό χρόνο.

- **EXPSPACE(Deterministic Exponential Space):** Σε αυτή την κλάση περιλαμβάνονται τα προβλήματα που μπορούν να επιλυθούν με εκθετική μεταβολή του χώρου.
- **R(Recursive):** Αυτή η κλάση περιλαμβάνει όλα τα προβλήματα που μπορούν να επιλυθούν σε πεπερασμένο χρόνο. Όσα προβλήματα δεν ανήκουν σε αυτή την κλάση απαιτούν άπειρους πόρους για να επιλυθούν και κατά συνέπεια δεν είναι επιλύσιμα. Ένα τέτοιο πρόβλημα είναι το πρόβλημα τερματισμού(Halting Problem).



https://sms.cam.ac.uk/image/546662

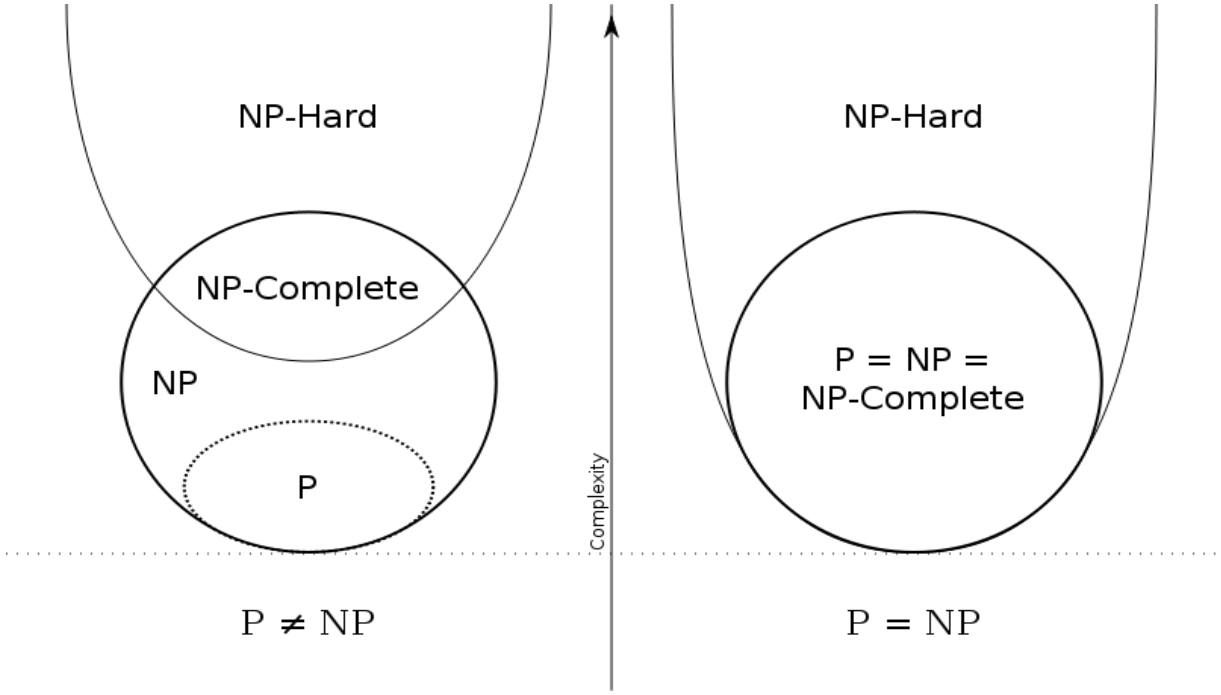
Η ταξινόμηση προβλημάτων βάσει της πολυπλοκότητας τους είναι πολύ σημαντική για τον κλάδο της βελτιστοποίησης. Τα προβλήματα που ανήκουν στην κλάση **P** μπορούν να επιλυθούν αποδοτικά από ακριβείς αλγορίθμους. Για όλες τις υπόλοιπες κλάσεις προβλημάτων χρησιμοποιούνται στοχαστικοί αλγόριθμοι που προσεγγίζουν την βέλτιστη λύση αλλά όχι πάντα με ακρίβεια. Από απλή επόπτευση του παραπάνω σχήματος εύκολα συμπεραίνεται ότι τα περισσότερα προβλήματα δεν μπορούν να επιλυθούν από ακριβείς αλγορίθμους. Βέβαια, τα προβλήματα της κλάσης **NP** δεν έχει αποδειχθεί ακόμα αν μπορούν να επιλυθούν ντετερμινιστικά.

P vs. NP

Με την έκφραση “P vs NP” περιγράφεται ένα από τα μεγαλύτερα άλυτα μυστήρια στην Πληροφορική και τα Μαθηματικά. Η επίσημη διατύπωση του προβλήματος είναι η εξής: *"Εάν κάθε πρόβλημα του οποίου η ύπαρξη λύσης μπορεί να επιβεβαιωθεί γρήγορα από έναν υπολογιστή μπορεί επίσης και να επιλυθεί γρήγορα από τον υπολογιστή;"*. Πρόκειται για ένα

από τα επτά προβλήματα για τα οποία δίνεται αμοιβή 1 εκατομμύριο δολάρια από το Clay Mathematics Institute για την πρώτη σωστή λύση.

Στην ουσία το ερώτημα που τίθεται είναι αν μπορεί να βρεθεί αλγόριθμος που να μπορεί να επιλύει ντετερμινιστικά και σε πολυωνυμικό χρόνο, προβλήματα της κλάσης NP. Αν αποδειχθεί ότι P = NP τότε ολόκληρη η κλάση NP θα καταρρεύσει όπως φαίνεται και στην παρακάτω εικόνα.



https://upload.wikimedia.org/wikipedia/commons/thumb/a/a0/P_np_np-complete_np-hard.svg/300px-P_np_np-complete_np-hard.svg.png

Οι συνέπειες μιας τέτοιας περίπτωσης(P = NP) θα είναι τεράστιες. Εάν βρεθεί ένας αλγόριθμος που να μπορεί να επιλύσει γρήγορα και αποτελεσματικά όλα τα προβλήματα της κλάσης NP τότε πάρα πολλοί κλάδοι θα αλλάξουν δραματικά. Σε πάρα πολλούς και διαφορετικούς κλάδους υπάρχουν NP–Complete προβλήματα που έχουν θεμελιώδη σημασία. Από την ημέρα που τέθηκε το ερώτημα έχουν γίνει πολλές απόπειρες να βρεθεί η απόδειξή του. Ωστόσο, ακόμα δεν υπάρχει μια αρκετά πειστική απόδειξη. Αξίζει να επισημανθεί πως υπάρχει διαφορά ανάμεσα στην εύρεση μιας απόδειξης που θα δείχνει ότι P = NP, και στην εύρεση ενός αλγορίθμου που θα μπορεί να λύσει NP-Complete προβλήματα. Από δημοσκοπήσεις που έχουν διεξαχθεί οι περισσότεροι ερευνητές πιστεύουν ότι P ≠ NP.

Ο Scott Aaronson, επιστήμονας της θεωρητικής Πληροφορικής είχε αναφέρει σχετικά: ***"Το P vs. NP πρόβλημα αποκαλείται ως 'ένα από τα μεγαλύτερα προβλήματα των σύγχρονων μαθηματικών και της θεωρητικής πληροφορικής'. Ο χαρακτηρισμός αυτός είναι υποτιμητικός. Το ερώτημα που τίθεται δεν είναι καθοριστικό μόνο για τον δικό μας κλάδο:***

είναι ένα από τα βαθύτερα ερωτήματα που τέθηκαν ποτέ και από το οποίο θα μπορούμε να επαληθεύουμε μια απάντηση. Η ικανότητα αναγνώρισης μιας λύσης συνεπάγεται και την ικανότητα εύρεσής της; Δεν αναζητούμε απλώς μια μαθηματική σχέση αλλά την φύση της μαθηματικής σκέψης καθεαυτής".

ΙΙΙ. ΕΥΡΙΣΤΙΚΟΙ ΜΕΘΟΔΟΙ

Επισκόπηση

Ευρετική ή Ευριστική(heuristic) μέθοδος ονομάζεται κάθε διαδικασία προσέγγισης της λύσης ενός προβλήματος η οποία όμως δεν εγγυάται ότι θα βρει την βέλτιστη λύση. Συνήθως οι ευρετικοί μέθοδοι αναζητούν κάποια λύση που θα είναι αρκετά ικανοποιητική σε προβλήματα που είναι δύσκολο ή και αδύνατο να επιλυθούν γρήγορα με ακρίβεια. Στην ουσία πρόκειται για τεχνικές που προσπαθούν να μαντέψουν "έξυπνα" μια λύση όταν ο συνολικός χώρος των πιθανών λύσεων, ή και των παραμέτρων του προβλήματος, είναι πολύ μεγάλος. Οι ευρετικές μέθοδοι βρίσκουν εφαρμογή τόσο στην καθημερινότητά όσο και στην επιστήμη της Πληροφορικής.

Στην Πληροφορική, χρήση ευρετικών μεθόδων γίνεται συνήθως όταν οι κλασσικές μέθοδοι βελτιστοποίησης αποτυγχάνουν να βρουν το βέλτιστο αποτέλεσμα ή όταν χρειάζονται πολύ χρόνο, ή και χώρο(μνήμη), για να δώσουν επαρκώς ικανοποιητικά αποτελέσματα. Με αυτό τον τρόπο θυσιάζεται η ακρίβεια και πληρότητα της λύσης ενός προβλήματος προκειμένου να επιλυθεί πιο γρήγορα. Συγκεκριμένα, μόνο λίγα προβλήματα μπορούν να επιλυθούν από κλασσικές μεθόδους βελτιστοποίησης(μη ευρετικές). Από τις γνωστές κλάσεις υπολογιστικής πολυπλοκότητας μονάχα στα προβλήματα της κλάσης **P(Polynomial Time)** δεν επιλέγονται ευρετικοί μέθοδοι.

Ο βασικός κορμός των ευρετικών μεθόδων βασίζεται σε κάποιον αλγόριθμο ο οποίος θα οργανώσει την αναζήτηση και θα ερευνήσει όσο το δυνατόν καλύτερα το πεδίο των λύσεων. Συνήθως εξαρτώνται και προσαρμόζονται στο πρόβλημα που πρόκειται να αντιμετωπίσουν.

Στην ίδια κατηγορία αλγορίθμων εντάσσονται και οι **μεταευρετικοί αλγόριθμοι(metaheuristics)**. Οι μεταευρετικοί αλγόριθμοι αποτελούν γενίκευση των ευρετικών. Στην ουσία πρόκειται για ένα υψηλότερο επίπεδο στρατηγικής το οποίο μπορεί να εφαρμοστεί σε διαφορετικά προβλήματα χωρίς να απαιτούνται ιδιαίτερες προσαρμογές. Οι χαμηλότερου επιπέδου διαδικασίες είναι συνήθως ευρετικοί μέθοδοι που μπορούν να ρυθμιστούν ειδικά για την επίλυση κάποιου προβλήματος. Ο μεταυρετικός αλγόριθμος χειρίζεται με αφηρημένο και γενικό τρόπο αυτές τις χαμηλότερου επιπέδου διαδικασίες χωρίς να προσαρμόζεται στο εκάστοτε πρόβλημα. Σήμερα μεταευρετικοί αλγόριθμοι χρησιμοποιούνται εκτενώς τόσο σε εμπορικό επίπεδο(βιομηχανίες) όσο και σε επίπεδο έρευνας(πανεπιστήμια). Συμβάλλουν καθοριστικά σε πολλές εφαρμογές και έχουν προσελκύσει το ενδιαφέρον πολλών ερευνητών καθώς έχει γίνει αντιληπτό ότι η σημασία τους είναι μεγάλη και αγγίζει πολλούς τομείς.

Μια ακόμα πιο γενική κατηγορία ευρετικών αλγορίθμων είναι οι **υπερευρετικοί αλγόριθμοι(hyperheuristics)**. Έχουν λίγο διαφορετική φιλοσοφία καθώς η αναζήτησή τους εστιάζει στην εύρεση του καταλληλότερου ευρετικού αλγορίθμου που θα βρει την λύση και όχι στην εύρεση της λύση καθεαυτής. Είναι ευρετικοί αλγόριθμοι που ψάχνουν ευρετικούς αλγόριθμους.

Σήμερα, υπάρχουν πολλοί διαφορετικοί μεταευρετικοί αλγόριθμοι. Παρότι, όλοι οι αλγόριθμοι έχουν την ίδια φιλοσοφία και μπορούν να επιλύσουν κάθε πρόβλημα βελτιστοποίησης δεν έχουν πάντα την ίδια απόδοση σε όλα τα προβλήματα. Οι μικρές διαφορές που έχουν στους μηχανισμούς τους, καθιστούν ιδιαίτερα κρίσιμη την επιλογή του κατάλληλου αλγορίθμου όταν πρόκειται να αντιμετωπιστεί ένα συγκεκριμένο πρόβλημα.

Ιστορία

➢ Προ-Θεωρητική Περίοδος(μέχρι 1940)

Ο άνθρωπος πάντοτε χρησιμοποιούσε ευριστικές μεθόδους για να λύσει προβλήματα της καθημερινότητας. Αυτή η ικανότητα ήταν πολύ σημαντική στο παρελθόν για να του δώσει περισσότερες πιθανότητες επιβίωσης έναντι άλλων ειδών. Και είναι ξεκάθαρο ότι ο ανθρώπινος νους λύνει πολλά από αυτά τα προβλήματα με προσεγγιστικό και όχι με ακριβή τρόπο. Για παράδειγμα, όταν έπρεπε να υπολογιστεί η τροχιά ενός ακοντίου για να πετύχει κάποιο άγριο θηρίο, δεν χρειαζόταν να υπολογιστεί η βέλτιστη τροχιά αλλά μια αρκετά καλή ώστε να πετύχει το στόχο του. Η ταχύτητα υπολογισμού είναι συχνά σημαντικότερη από την ακρίβεια. Πολλές φορές αυτό γίνεται υποσυνείδητα και χωρίς να χρειάζεται εκπαίδευση. Αυτό αποδεικνύει ότι ο ανθρώπινος εγκέφαλος είναι βιολογικά εξοπλισμένος για να λύνει ένα τεράστιο εύρος από προβλήματα, πολλά εκ των οποίων θα μπορούσαν να χαρακτηριστούν και προβλήματα βελτιστοποίησης. Παρά το γεγονός όμως ότι οι ευριστικοί μέθοδοι είναι εντελώς φυσικοί στον άνθρωπο, η επιστημονική μελέτη τους ξεκίνησε πολύ αργότερα.

➢ Πρώιμη Εποχή(από 1940 μέχρι 1980)

Λίγο μετά τον Β’ Παγκόσμιο Πόλεμο ο Ούγγρος μαθηματικός **George Polya(1887 - 1985)** δημοσίευσε έναν μικρό τόμο στον οποίο πρότεινε την επίλυση προβλημάτων μέσα από γενικότερες και υψηλότερου επιπέδου στρατηγικές. Στρατηγικές δηλαδή, που μπορούν να εφαρμοστούν σε μεγάλο εύρος προβλημάτων αντλώντας έμπνευση από λύσεις που υπάρχουν σε καθημερινά προβλήματα. Ο Polya δεν πρότεινε κάποιον αλγόριθμο ούτε έλυσε κάποιο πρόβλημα. Έθεσε όμως τα θεμέλια των μετέπειτα μεταευρετικών αλγορίθμων. Η αρχή της επαγωγής και η εξαγωγή συμπερασμάτων μέσα από την μελέτη της πραγματικής ζωής, ήταν μια ιδέα που αποδείχθηκε

καθοριστική για την εξέλιξη των μεταευριστικών αλγορίθμων. Παράλληλα, την ίδια εποχή αναπτύχθηκαν κάποιες ευρετικές τεχνικές όπως ο *Άπληστος Αλγόριθμος* ή ο *Αλγόριθμος του Dijkstra*. Τεχνικές οι οποίες χρησιμοποιούνται ακόμα και σήμερα.

- **Εποχή των Μεθόδων(1980 – 2000)**

Παρόλο που νωρίτερα είχαν αναπτυχθεί ιδέες για μεταευρετικές μεθόδους εντούτοις δεν υπήρχαν ακόμα αρκετά καλοί αλγόριθμοι ή υλοποιήσεις αυτών των ιδεών. Αργότερα όμως, και κυρίως με την εμφάνιση των εξελικτικών αλγορίθμων ολόκληρος ο τομέας άρχισε να συγκεντρώνει το ενδιαφέρον των ερευνητών. Αυτό έγινε σταδιακά από τις αρχές της δεκαετίας του ‘60 μέχρι και το 1989. Οι ευριστικοί αλγόριθμοι που ήταν εμπνευσμένοι από την Εξέλιξη ήταν εξαιρετικά αποτελεσματικοί αλλά πέραν αυτού άνοιξαν το δρόμο για την ανάπτυξη μιας ολόκληρης οικογένειας αλγορίθμων που βασίζονται σε αλληγορίες. Αλληγορίες που είναι εμπνευσμένες είτε από τη φύση είτε από τον άνθρωπο. Η ανάπτυξη μεγάλου πλήθους αλγορίθμων είχε ως συνέπεια την ανακάλυψη πολλών νέων μηχανισμών βελτιστοποίησης. Την ίδια περίοδο, πολλοί ερευνητές πίστεψαν ότι θα ήταν δυνατό να δημιουργηθεί κάποιο αλγοριθμικό framework που θα μπορεί να αντιμετωπίζει αποτελεσματικά κάθε πρόβλημα χωρίς να τροποποιείται.

- **Εποχή των Frameworks(2000 – σήμερα)**

Σ’ αυτή τη περίοδο προτάθηκαν για πρώτη φορά οι υβριδικοί μεταευρετικοί. Πρόκειται για μίξη μηχανισμών, που προέρχονται από διαφορετικούς μεταευρετικούς αλγορίθμους, ή μίξη μεταευρετικών με ακριβείς μεθόδους βελτιστοποίησης. Ένα τέτοιο παράδειγμα είναι οι Μιμητικοί Αλγόριθμοι(Memetic Algorithm). Επιπλέον, την ίδια περίοδο άρχισε να υπάρχει έντονη αμφισβήτηση και κριτική για τους αλγορίθμους που βασίζονται σε κάποια αλληγορία. Το βασικό επιχείρημα ήταν ότι δημιουργούνται συνεχώς νέοι αλγόριθμοι χωρίς όμως να προσφέρουν κάτι ουσιαστικό στον κλάδο καθώς αποτελούν παραλλαγή αλγορίθμων και μηχανισμών που ήδη υπάρχουν. Σήμερα, μεταευρετικοί μέθοδοι χρησιμοποιούνται ευρέως και λύνουν αποτελεσματικά προβλήματα της πραγματικής ζωής. Ακόμα όμως του λείπει η μαθηματική θεμελίωση και αντιμετωπίζονται από πολλούς ως τέχνη.

- **Επιστημονική Εποχή(μέλλον)**

Για πολύ μεγάλο διάστημα ο κλάδος των μεταευρετικών δεν αντιμετωπιζόταν με την δέουσα προσοχή από την ερευνητική κοινότητα. Πολλοί ερευνητές τους υποτιμούσαν διότι τους έλειπε το μαθηματικό υπόβαθρο. Ενώ αντίθετα οι ακριβείς μέθοδοι είχαν

θεμελιωθεί από την πρώτη στιγμή με μαθηματικό τρόπο. Οι πρώτες προσπάθειες που είχαν γίνει για μπουν τα θεμέλια δεν ήταν ιδιαίτερα επιτυχείς. Η κατανόηση των μεταευρετικών συστημάτων σε θεμελιακό επίπεδο όσο δύσκολη είναι άλλο τόσο σημαντική είναι. Το γεγονός όμως ότι οι μεταευρετικοί μέθοδοι εφαρμόζονται σε πολλούς τομείς σημαίνει ότι είναι πολύ πιθανό στο μέλλον ο κλάδος να επεκταθεί και να αναπτυχθεί πολύ περισσότερο. Η έρευνες που θα γίνονται θα εστιάζουν περισσότερο στο μαθηματικό υπόβαθρο.

Χαρακτηριστικά

- **Εύρος Αναζήτησης**

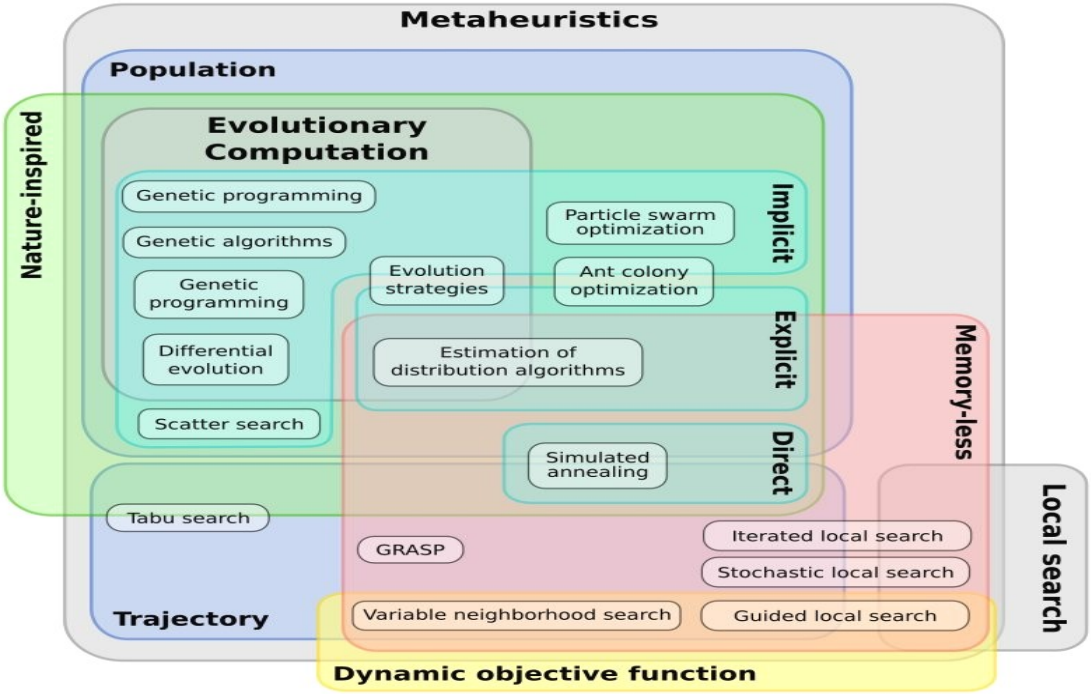
Υπάρχουν αλγόριθμοι που αναζητούν την βέλτιστη λύση τοπικά και αλγόριθμοι που αναζητούν την συνολικά βέλτιστη λύση. Συνήθως όταν γίνεται αναζήτηση τοπικά τότε υπάρχει μια λύση η οποία παραλλάσσεται ελαφρώς προς κάποια κατεύθυνση προκειμένου να εξερευνηθούν οι γειτονικές περιοχές. Αντίθετα, όταν αναζητείται η συνολικά βέλτιστη λύση τότε αναζητείται ολόκληρο το πεδίο ορισμού. Πολλές φορές αυτές οι δύο προσεγγίσεις συνδυάζονται όταν χρησιμοποιούνται Υβριδικοί Αλγόριθμοι(Hybrid Algorithms).

- **Μνήμη**

Ένα σημαντικό χαρακτηριστικό των Μεταευρετικών Αλγορίθμων είναι η Μνήμη. Είναι η δυνατότητα που έχει ένας μεταευρετικός να αποκτά εμπειρία κατά τη διάρκεια της αναζήτησης. Η αποκτώμενη εμπειρία είναι εξαιρετικά χρήσιμη, καθώς αποτρέπει τον εκάστοτε αλγόριθμο από την αναζήτηση περιοχών που είναι απίθανο να δώσουν χρήσιμες πληροφορίες. Ή αντίθετα αν κάποια περιοχή είναι πλούσια σε πληροφορίες τότε αναζητείται περισσότερο. Αυτό το χαρακτηριστικό χρησιμοποιείται εκτενώς στην μέθοδο Tabu Search. Για την ακρίβεια, οι περισσότεροι μεταευρετικοί κάνουν χρήση αυτού του μηχανισμού. Οι αλγόριθμοι που δεν χρησιμοποιούν μνήμη είναι συνήθως καλοί στην τοπική αναζήτηση. Τέτοιοι αλγόριθμοι είναι οι Simulated Annealing, GRASP κ.α..

➢ Πληθυσμός Λύσεων

Κάποιοι μεταευρετικοί χρησιμοποιούν έναν πληθυσμό από υποψήφιες λύσεις ενώ υπάρχουν και μεταευρετικοί που σε όλη τη διάρκεια της αναζήτησης χρησιμοποιούν μια μόνο λύση. Όταν υπάρχει πληθυσμός λύσεων τότε οι υπάρχει η δυνατότητα οι υποψήφιες λύσεις να ανταλλάζουν πληροφορίες μεταξύ τους, να ανασυνδυάζονται και να εξάγουν χρήσιμα συμπεράσματα. Μερικοί τέτοιοι αλγόριθμοι είναι οι γενετικοί αλγόριθμοι, η βελτιστοποίηση αποικίας μυρμηγκιών κ.α.. Στην περίπτωση που ο μεταυρετικός δεν χρησιμοποιεί πληθυσμό λύσεων τότε υπάρχει μία και μοναδική λύση η οποία τροποποιείται συνεχώς και οδηγείται προοδευτικά προς το βέλτιστο.



https://upload.wikimedia.org/wikipedia/commons/thumb/c/c3/Metaheuristics_classification.svg/220px-Metaheuristics_classification.svg.png

➢ Συμβολισμοί

Οι περισσότεροι από τους μεταευρετικούς αλγορίθμους βασίζονται σε κάποια αλληγορία που είναι εμπνευσμένη από την πραγματική ζωή και από φυσικές διαδικασίες. Εξέλιξη, αποικίες μυρμηγκιών και μελισσών, μουσική, μεταλλουργία, σταγόνες νερού και πολλές άλλες παρόμοιες έννοιες υπήρξαν αφορμή για να γεννηθούν πολλοί μεταευρετικοί αλγόριθμοι. Υπάρχουν πολλοί ερευνητές όμως που έχουν αρχίσει να αμφισβητούν τις νεότερες από αυτές τις μεθόδους υποστηρίζοντας ότι δεν συνεισφέρουν κάτι ουσιαστικό στον κλάδο.

No Free Lunch

Με την φράση “No Free Lunch”(μη δωρεάν γεύματα) είναι γνωστό το θεώρημα των David Wolpert και William G. Macready. Η φράση “No free lunch” οφείλεται στον συγγραφέα Robert Heinlein και συνοψίζει την ιδέα πως όλα τα πράγματα στον κόσμο έχουν ένα κόστος, ακόμα κι αν αυτό είναι καλά κρυμμένο.

Το θεώρημα αποδεικνύει ότι σε προβλήματα βελτιστοποίησης, το μέσο υπολογιστικό κόστος επίλυσής όλων των προβλημάτων μιας υπολογιστικής κλάσης, είναι ίδιο για όλες τις μεθόδους επίλυσης. Αν ένας αλγόριθμος Α αποδίδει καλύτερα από έναν αλγόριθμο Β σε κάποια προβλήματα τότε υπάρχει ένα ίσο πλήθος προβλημάτων που ο αλγόριθμος Β αποδίδει καλύτερα από τον αλγόριθμο Α. Με άλλα λόγια, είναι αδύνατο να βρεθεί αλγόριθμος καθολικής βελτιστοποίησης(global optimization) που να είναι ο καλύτερος σε όλες τις αντικειμενικές συναρτήσεις. Το No free lunch θεώρημα έθεσε τα όρια στους στοχαστικούς μεταευρετικούς αλγορίθμους.

Ωστόσο, έχουν γίνει έρευνες που δείχνουν ότι μπορούν να υπάρξουν “δωρεάν γεύματα” σε μεθόδους **Υπολογιστικής Συνεξέλιξης(Coevolutionary Computation)**. Οι έρευνες αυτές, εστιάζουν σε συγκεκριμένες κατηγορίες προβλημάτων όπου η λύση προσεγγίζεται συνδυάζοντας τους κλάδους της Εξελικτικής Υπολογιστικής και της Θεωρίας Παιγνίων. Σε αυτά τα προβλήματα, ένα σύνολο παικτών(υποψήφιων λύσεων) συνεργάζονται και παράγουν μαζί έναν νικητή. Στη συνέχεια, αυτός ο νικητής επιλέγει έναν ή περισσότερους παίκτες ως ανταγωνιστές για το επόμενο παιχνίδι. Στόχος κάθε φορά είναι η ανάδειξη ενός καλού παίκτη χωρίς όμως να υπάρχει αντικειμενική συνάρτηση. Η ποιότητα ενός παίκτη εκτιμάται από το πόσο καλά αποδίδει κόντρα στους ανταγωνιστές του. Έτσι, ένας αλγόριθμος μπορεί να χρησιμοποιεί την ποιότητα των παικτών και να οδηγείται στην επιλογή παικτών με ολοένα και καλύτερη ποιότητα.

IV. ΣΥΜΒΟΛΙΚΟΙ ΜΕΤΑΥΡΕΤΙΚΟΙ ΑΛΓΟΡΙΘΜΟΙ

Γενετικός Αλγόριθμος

- **Επισκόπηση**

Οι γενετικοί Αλγόριθμοι(Genetic Algorithms) είναι προσαρμοστικοί αλγόριθμοι και αναζητούν το συνολικό βέλτιστο. Ανήκουν στην ευρύτερη μελέτη της Εξελικτικής Υπολογιστικής(Evolutionary Computation) και ίσως είναι ο πιο δημοφιλής μεταευρετικός αλγόριθμος.

- **Συμβολισμός**

Ο γενετικός αλγόριθμος εμπνεύστηκε από την βιολογική Εξέλιξη των ειδών. Άτομα του πληθυσμού συνεισφέρουν το γενετικό τους υλικό(γονότυπος) στην επόμενη γενιά(απόγονοι) αναλογικά με την προσαρμοστικότητα που έχουν στο περιβάλλον. Κάθε επόμενη γενιά δημιουργείται μέσα από το ζευγάρωμα ατόμων του πληθυσμού. Το ζευγάρωμα περιλαμβάνει διαδικασίες όπως είναι ο ανασυνδυασμός γονιδίων και η μετάλλαξη(πιθανότητα λάθους).

- **Ψευδοκώδικας**

```
Αρχικοποίησε και αξιολόγησε των πληθυσμό των ατόμων.
While (κριτήριο τερματισμού) Do
   Επέλεξε δύο άτομα του πληθυσμού για αναπαραγωγή.
   Δημιούργησε δύο νέα άτομα με χρήση τελεστών
αναπαραγωγής.
   Αξιολόγησε τα νέα άτομα που δημιουργήθηκαν.
   Συμπλήρωσε την επόμενη γενιά του πληθυσμού.
End While
```

Μιμητικός Αλγόριθμος

- **Επισκόπηση**

Ο Μιμητικός Αλγόριθμος(Memetic Algorithm) είναι μια ακόμη Μεταευρετική μέθοδος που προτάθηκε για πρώτη φορά από τον Pablo Moscato το 1989. Και αυτή η μέθοδος εμπνεύστηκε από την εξέλιξη των ειδών ωστόσο παρουσιάζει κάποιες διαφορές σε σχέση με τον Γενετικό Αλγόριθμο. Η μέθοδος εστιάζει κυρίως στην εξέλιξη ενός ατόμου του πληθυσμού παράλληλα με την εξέλιξη του συνόλου του πληθυσμού.

- **Συμβολισμός**

Ο γονότυπος εξελίσσεται μέσα από την αλληλεπίδραση που έχει ο φαινότυπος με το περιβάλλον. Η αλληλεπίδραση αυτή σχετίζεται με την κουλτούρα που αποκτά ένα άτομο από τον πολιτισμό της κοινωνίας που βρίσκεται. Η κουλτούρα μπορεί να επηρεάσει τους μηχανισμούς επιλογής της εξέλιξης ή ακόμα και τους μηχανισμούς ζευγαρώματος. Αυτές οι "πολιτισμικές πληροφορίες" διαδίδονται από άτομο σε άτομο και κατά συνέπεια μια ολόκληρη κοινωνία οδηγεί την εξέλιξη με τον πολιτισμό της.

- **Ψευδοκώδικας**

```
Αρχικοποίησε τον πληθυσμό των ατόμων.
While (κριτήριο επαναλήψεων) Do
      Αξιολόγησε όλα τα άτομα του πληθυσμού.
      Εξέλιξε τον πληθυσμό με χρήση γενετικών τελεστών.
      Επέλεξε ένα υποσύνολο ατόμων Ω.
      For (όλα τα άτομα του πληθυσμού Ω) Do
            Εξέλιξε τον πληθυσμό Ω με χρήση τελεστών
            τοπικής αναζήτησης.
      End For
End While
```

Προσομοιωμένη Ανόπτηση

- **Επισκόπηση**

Η προσομοιωμένη ανόπτηση(Simulated Annealing) είναι ένας στοχαστικός μεταυρετικός αλγόριθμος βελτιστοποίησης και δεν χρησιμοποιεί το χαρακτηριστικό της μνήμης ωστόσο υπάρχουν πολλές παραλλαγές του αλγορίθμου.

- **Συμβολισμός**

Ο αλγόριθμος εμπνεύστηκε από διαδικασίες που χρησιμοποιούνται στην μεταλλουργία για την πυράκτωση και τη δημιουργία μεταλλικών εργαλείων. Αρχικά, το μέταλλο θερμαίνεται και στη συνέχεια πρέπει να ψυχρανθεί αργά και σταθερά ώστε να αυξηθεί το μέγεθος των κρυστάλλων στο υλικό και να είναι πιο ανθεκτικό. Κάθε διαμόρφωση μιας λύσης στο χώρο αναζήτησης αντιπροσωπεύει και μια διαφορετική εσωτερική ενέργεια του μετάλλου. Η θέρμανση του συστήματος έχει ως αποτέλεσμα την χαλάρωση των κριτηρίων αποδοχής των λύσεων που ελήφθησαν από το χώρο αναζήτησης. Καθώς το σύστημα ψύχεται, τα κριτήρια αποδοχής των λύσεων περιορίζονται και επικεντρώνονται στη βελτίωση των κινήσεων. Μόλις το μέταλλο κρυώσει, η διαμόρφωση θα αντιπροσωπεύει μια αρκετά καλή λύση.

- **Ψευδοκώδικας**

```
Αρχικοποίησε και αξιολόγησε μια λύση.
While (κριτήριο τερματισμού) Do
   Όρισε περιοχές γειτονικών λύσεων.
   Επέλεξε μια γειτονική λύση με χρήση στοχαστικών
τελεστών.
   Αντικατέστησε την αρχική λύση με πιθανότητα ανάλογη
της ποιότητας της.
End While
```

Βελτιστοποίηση Αποικίας Μυρμηγκιών

- **Επισκόπηση**

Η βελτιστοποίηση αποικίας μυρμηγκιών(Ant colony Optimization) προτάθηκε για πρώτη φορά από τον Marco Dorigo στην διδακτορική του διατριβή το 1992. Ανήκει στην οικογένεια των αλγορίθμων Νοημοσύνης Σμήνους(Swarm Intelligence). Ο αλγόριθμος αρχικά σχεδιάστηκε για να βρίσκει τη βέλτιστη διαδρομή σε γράφους. Πολύ γρήγορα, έγινε αντιληπτό ότι μπορούσε να εφαρμοστεί σε πολύ περισσότερα προβλήματα βελτιστοποίησης.

- **Συμβολισμός**

Όταν τα μυρμήγκια αναζητούν τροφή περιπλανιούνται τυχαία στο περιβάλλον τους μέχρι να βρουν τροφή. Μόλις κάποιο μυρμήγκι εντοπίσει τροφή τότε απελευθερώνει μια χημική ουσία που λέγεται φερομόνη. Οι πιθανές διαδρομές ανάμεσα στη τροφή και την αποικία είναι πάρα πολλές. Οπότε όταν κάποια από αυτές τις διαδρομές ακολουθηθεί πολλές φορές από κάποιο μυρμήγκι τότε θα έχει περισσότερη φερομόνη και κατά συνέπεια είναι πιθανότερο να ακολουθηθεί από τα υπόλοιπα μυρμήγκια.

- **Ψευδοκώδικας**

```
Αρχικοποίησε και αξιολόγησε τον πληθυσμό των μυρμηγκιών.
While (κριτήριο τερματισμού) Do
     Τοποθέτησε το κάθε μυρμήγκι σε έναν αρχικό κόμβο.
     For (κάθε μυρμήγκι του πληθυσμού) Do
          Επέλεξε τον επόμενο κόμβο.
          Ενημέρωσε την φερομόνη.
     End For
     Ενημέρωσε την καλύτερη λύση.
     Ενημέρωσε την φερομόνη.
End While
```

Αλγόριθμος Μελισσών

➢ Επισκόπηση

Ο αλγόριθμος μελισσών(Honey Bee Algorithm) είναι μια ακόμη μεταευρετική μέθοδος που ανήκει στο πεδίο των αλγορίθμων Νοημοσύνης Σμήνους. Αναπτύχθηκε για πρώτη φορά το 2005 από τον Dervis Karaboga. Σαν μέθοδος έχει αρκετά κοινά στοιχεία με την Βελτιστοποίηση Αποικίας Μυρμηγκιών.

➢ Συμβολισμός

Η μέθοδος γεννήθηκε μέσα από την μελέτη της συλλογικής συμπεριφοράς των μελισσών. Οι μέλισσες συλλέγουν νέκταρ από τα λουλούδια ως πηγή τροφής για την κυψέλη τους. Αρχικά, στέλνονται από την κυψέλη μέλισσες - ανιχνευτές για να εντοπίσουν περιοχές λουλουδιών που είναι πλούσιες σε νέκταρ. Κατόπιν, ενημερώνουν τις υπόλοιπες μέλισσες για την ποιότητα και την τοποθεσία αυτών των λουλουδιών μέσω του αποκαλούμενου “Χορού των Μελισσών”. Κάθε φορά η μέλισσα-ανιχνευτής επιστρέφει στα λουλούδια με περισσότερες μέλισσες οι οποίες εξακολουθούν να ενημερώνουν τη κυψέλη για την ποιότητα του νέκταρ. Παράλληλα, στέλνονται νέοι ανιχνευτές που εξακολουθούν να αναζητούν περιοχές που μπορεί να είναι πλούσιες σε νέκταρ.

➢ Ψευδοκώδικας

```
Αρχικοποίησε και αξιολόγησε τον πληθυσμό των μελισσών.
Αρχικοποίησε και αξιολόγησε τις περιοχές γης.
While (κριτήριο τερματισμού) Do
      Επέλεξε περιοχές για τοπική αναζήτηση.
      Ανέθεσε τις περισσότερες μέλισσες στις περιοχές
      τοπικής αναζήτησης.
      Επέλεξε τις καλύτερες περιοχές.
      Ανέθεσε τις υπόλοιπες μέλισσες για τυχαία αναζήτηση.
End While
```

Συνοπτικός Πίνακας Κυριότερων Συμβολικών Μεταευρετικών

Μέθοδος	Αγγλική Ονομασία	Έτος	Δημιουργός	Έμπνευση
Γενετικός Αλγόριθμος	Genetic Algorithm	1975	John Henry Holland	Εξέλιξη
Μιμητικός Αλγόριθμος	Memetic Algorithm	1989	Pablo Moscato	Μιμίδια
Προσομοιωμένη Ανόπτηση	Simulated Annealing	1983	Scott Kirkpatrick	Μεταλλουργία
Βελτιστοποίηση Αποικίας Μυρμηγκιών	Ant Colony Optimization	1992	Marco Dorigo	Συμπεριφορά Μυρμηγκιών
Αλγόριθμος Μελισσών	Honey Bees Algorithm	2004	Sunil Nakrani & Craig Tovey	Συμπεριφορά Μελισσών
Αναζήτηση Αρμονίας	Harmony Search	2001	Zong Woo Geem	Μουσική

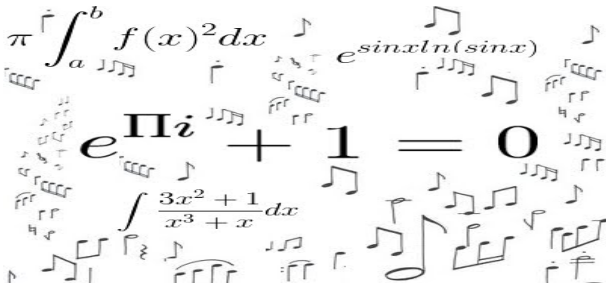
V. Αλγόριθμος Αναζήτησης Αρμονίας

Επισκόπηση

Ο **Αλγόριθμος Αναζήτησης της Αρμονίας(Harmony Search Algorithm)** είναι ένας μεταευρετικός αλγόριθμος που είναι εμπνευσμένος από την μουσική. Προτάθηκε για πρώτη φορά επίσημα από τους Zong Woo Geem, Joong Hoon Kim και Gobichettipalayam Vasudevan Loganathan το 2001. Ένα χρόνο νωρίτερα(2000) όμως ο Zong Woo Geem είχε ήδη χρησιμοποιήσει για πρώτη φορά τον αλγόριθμο σε έρευνα που είχε πραγματοποιήσει και είχε ως θέμα την βέλτιστη σχεδίαση υδραυλικών δικτύων. Πολύ γρήγορα ο αλγόριθμος άρχισε να γίνεται γνωστός και να επεκτείνεται σε πολλούς κλάδους(βιομηχανία, ιατρική, συστήματα ελέγχου κ.α.).

Μαθηματικά και Μουσική

Τα Μαθηματικά και η Μουσική είναι δυο κλάδοι που έχουν πολύ μεγάλη σχέση μεταξύ τους. Από πολύ παλιά ο άνθρωπος διαπίστωσε και μελέτησε την αλληλεπίδραση που έχουν οι δύο τέχνες.



http://98five.com/wp-content/uploads/2016/03/maths-and-music.jpg

Στην Αρχαιότητα, οι Πυθαγόρειοι ήταν οι πρώτοι που ανακάλυψαν αυτή τη σχέση. Οι Πυθαγόρειοι ήταν μια μυστικιστική οργάνωση που είχαν στο επίκεντρο της ιδεολογίας τους τα μαθηματικά. Ιδρύθηκαν από τον Πυθαγόρα τον Σάμιο γύρω στον 6ο αιώνα π.Χ.. Η μελέτη της αλληλεπίδρασης των μαθηματικών και της μουσικής ξεκίνησε από τον ίδιο τον Πυθαγόρα και συνεχίστηκε από την οργάνωση. Ιδιαίτερα σημαντική είναι η προσφορά του Αρχύτα και του Φιλόλαου οι οποίοι ήταν και αυτοί Πυθαγόρειοι. Συγκεκριμένα, οι Πυθαγόρειοι χρησιμοποιούσαν το μονόχορδο(μουσικό όργανο της εποχής, αποκαλούνταν και "Πυθαγόρειος κανών" διότι απέδιδαν την εφεύρεσή του στον Πυθαγόρα) για να καθορίσουν

τις μαθηματικές σχέσεις που διέπουν τους μουσικούς ήχους. Χρησιμοποιώντας το μονόχορδο παρατήρησαν ότι μόνο ακριβείς μαθηματικές σχέσεις έδιναν αρμονικούς ήχους. Για παράδειγμα, έπρεπε να χωριστεί ακριβώς στη μέση η χορδή, και όχι περίπου στη μέση, ώστε να παραχθεί ένας ευχάριστος αρμονικός ήχος. Αν μειωθεί το μήκος μιας χορδής ακριβώς στο μισό, τότε ο ήχος που παράγεται είναι ακριβώς μία οκτάβα υψηλότερος(μία οκτάβα είναι ένα ντο, ρε, μι, φα, σολ, λα, σι, ντο). Αν μειωθεί το μήκος της χορδής κατά 1/3, τότε τα 2/3 της χορδής που απομένουν δίνουν τη διαφορά της πέμπτης (δηλαδή από το ντο στο λα). Κι αν μειωθεί το μήκος κατά 1/4, τότε τα 3/4 που απομένουν δίνουν τη διαφορά της τετάρτης (από το ντο στο σολ). Ήταν ξεκάθαρο σ' αυτό το επίπεδο της παρατήρησης ότι τα μαθηματικά κυβερνούν τη μουσική. Το γεγονός ότι από τους ήχους αυτών των διαφορών δημιουργείται ένα ευχάριστο συναίσθημα στον ακροατή, οδήγησε τους Πυθαγορείους στο συμπέρασμα ότι οι αριθμοί ελέγχουν όχι μόνο τον άψυχο αλλά και τον έμψυχο κόσμο μέσω της μουσικής.

Ο **Ιάννης Ξενάκης(1922 – 2001)**, ένας από τους μεγαλύτερους Έλληνες συνθέτες και μηχανικούς χρησιμοποιούσε μαθηματικά μοντέλα για να συνθέσει μουσική. Χαρακτήριζε τα έργα του με τον όρο "Στοχαστική Μουσική" διότι χρησιμοποιούσε στοιχεία της Θεωρίας Πιθανοτήτων στα περισσότερα από τα έργα του. Ιδιαίτερη σημασία είχε και η Θεωρία Παιγνίων με βάση την οποία ο Ξενάκης δημιουργούσε έργα(που απαρτίζονταν από 2 μαέστρους) όπου ένας μαέστρος αντιδρά στις επιλογές που κάνει ο άλλος. Σε τέτοια έργα σημαντικό ρόλο έπαιζε η αυτοσχεδίαση. Παράλληλα, με τον όρο "Συμβολική Μουσική" χαρακτήριζε έργα τα οποία δημιουργούσε συνδυάζοντας στοιχεία της Άλγεβρας Μπουλ και της Θεωρίας Συνόλων.

Από την Αρχαιότητα μέχρι και την σύγχρονη εποχή η αλληλεπίδραση μαθηματικών και μουσικής έμεινε ζωντανή και εξακολουθεί ακόμα και σήμερα να αποτελεί αντικείμενο μελέτης. Είναι παραπάνω από προφανές ότι η σχέση των δύο κλάδων είναι πραγματική.

Αρμονία

Στη μουσική, ο όρος **αρμονία(harmony)** αναφέρεται στο ηχητικό αποτέλεσμα που παράγεται από την χρήση δύο ή περισσότερων μουσικών οργάνων. Από μια άλλη, πιο επιστημονική, οπτική γωνία ορίζεται ως η σχέση και η αλληλεπίδραση που έχουν δύο διαφορετικά ηχητικά κύματα που παράγονται από μουσικά όργανα ή και από την ανθρώπινη φωνή. Αυτή η αλληλεπίδραση είναι που καθορίζει αν το τελικό αποτέλεσμα, δηλαδή ο ήχος που παράγεται, ηχεί ευχάριστα.

Όπως σε κάθε τέχνη έτσι και στη μουσική οι έννοιες του καλαίσθητου και του ωραίου είναι υποκειμενικές και διαφέρουν από άνθρωπο σε άνθρωπο. Πολλοί παράγοντες επηρεάζουν το γούστο των ανθρώπων στη μουσική όπως ο κοινωνικός περίγυρος, ο πολιτισμός, τα παιδικά ερεθίσματα και πολλά άλλα. Ωστόσο έχει παρατηρηθεί πως στη μουσική υπάρχουν κάποιοι αντικειμενικοί κανόνες και κάποια μοτίβα που ισχύουν για κάθε είδος μουσικής. Φυσικά πολλοί από αυτούς τους κανόνες μπορεί να εξελίσσονται με το πέρασμα του χρόνου και να

προσαρμόζονται στο καλλιτεχνικό ρεύμα της εκάστοτε εποχής, όμως κάποιες βασικές αρχές παραμένουν αναλλοίωτες.

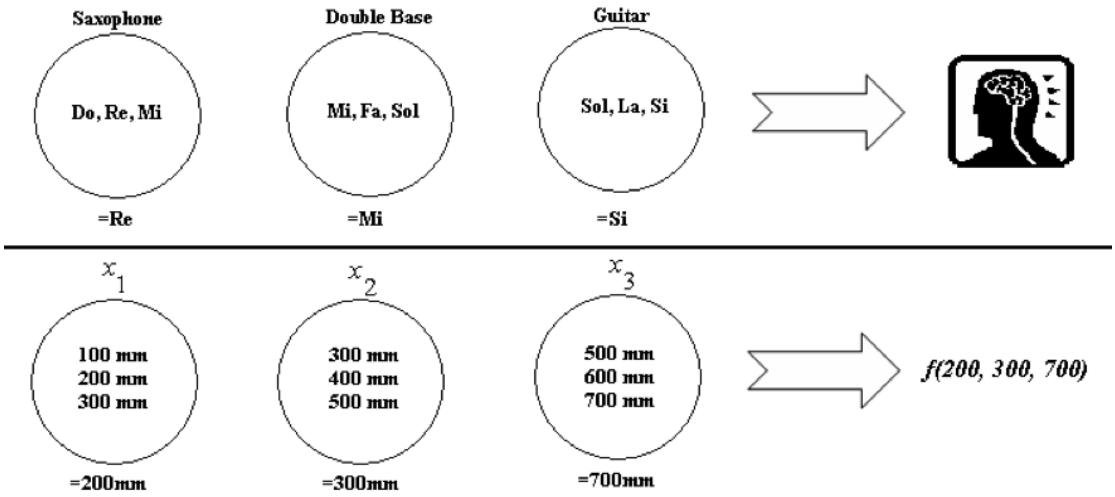
Μέσα σε ένα μουσικό σχήμα ένας μουσικός μπορεί να κάνει κάποια από τις παρακάτω επιλογές:

- Να παίξει κάποιο γνωστό κομμάτι. Ένα κομμάτι μουσικής δηλαδή που το γνωρίζει ήδη και μπορεί να το παίξει χωρίς δυσκολία.
- Να παίξει κάποια παραλλαγή κάποιου γνωστού κομματιού μεταβάλλοντας το ελαφρώς.
- Να κάνει κάποιον αυτοσχεδιασμό. Να παίξει δηλαδή κάποιο τελείως καινούργιο κομμάτι που δεν το ξέρει και που φυσικά δεν γνωρίζει αν το αποτέλεσμα θα είναι καλό ή κακό.

Αυτή είναι περίπου η διαδικασία με την οποία ο μουσικός αναζητεί την αρμονία ανεξάρτητα από το είδος μουσικής. Η επαναλαμβανόμενη παραγωγή κομματιών και η διαρκής προσαρμογή τους είναι μια διαδικασία βελτιστοποίησης. Είναι η αναζήτηση της βέλτιστης αρμονίας. Κάπως έτσι εμπνεύστηκε τον αλγόριθμο ο Zong Woo Geem.

Συμβολισμός

Η αναλογία ανάμεσα στην αυτοσχεδίαση μουσικής και την βελτιστοποίηση στη μηχανική φαίνεται στην παρακάτω εικόνα.



Alia, O., Mandava, R., "The variants of the harmony search algorithm: an overview", (2011)

Κάθε μουσικός(σαξοφωνίστας, μπασίστας, πιανίστας κλπ.) αντιστοιχεί σε μια μεταβλητή απόφασης(x_1, x_2, x_3) και το εύρος κάθε μουσικού οργάνου(σαξόφωνο{Ντο, Ρε, Μι}, μπάσο{Μι, Φα, Σολ}, κιθάρα{Σολ, Λα, Σι}) αντιστοιχεί στο πεδίο ορισμού της κάθε μεταβλητής απόφασης($x_1 \epsilon \{100, 200, 300\}$, $x_2 \epsilon \{300, 400, 500\}$, $x_3 \epsilon \{600, 700, 800\}$). Οπότε αν ο σαξοφωνίστας παίξει την νότα Ρε, ο μπασίστας την νότα Μι και ο κιθαρίστας την νότα Σι τότε η νέα Αρμονία αποτελείται από τις νότες Ρε-Μι-Σι. Αν η αρμονία αυτή έχει καλή αισθητική τότε απομνημονεύεται. Κατ' αναλογία αν το καινούργιο διάνυσμα(έστω ότι επιλέγονται οι τιμές 200-300-700) είναι καλό(με βάση την αντικειμενική συνάρτηση) τότε αποθηκεύεται. Έτσι, μέσα από πολλές συνεχόμενες επαναλήψεις η αρμονία βελτιώνεται και οδηγείται σταδιακά προς το βέλτιστο.

Αντιστοίχηση Εννοιών

- Μουσικό Όργανο/Νότα ⟶ Μεταβλητή Απόφασης
- Εύρος Τόνου ⟶ Πεδίο Ορισμού
- Αρμονία ⟶ Λύση
- Αισθητική ⟶ Αντικειμενική Συνάρτηση
- Εξάσκηση ⟶ Επανάληψη
- Εμπειρία ⟶ Μνήμη

Ορισμοί

- **Αρμονία(Harmony)**
  Κάθε σύνολο τιμών που αποτελείται από τις τιμές των μεταβλητών της αντικειμενικής συνάρτησης. Κάθε τέτοιο σύνολο τιμών είναι και μια υποψήφια λύση.
- **Πλήθος Αυτοσχεδιασμών(Number of Improvisations)**
  Είναι ο συνολικός αριθμός των επαναλήψεων. Εναλλακτικά, μπορεί να αντικατασταθεί από κάποιο άλλο κριτήριο τερματισμού.
- **Αρμονική Μνήμη(Harmony Memory)**
  Κατά τη διάρκεια της αναζήτησης αποθηκεύεται στη μνήμη του υπολογιστή ένα πλήθος από αρμονίες, δηλαδή δυνατές λύσεις.

- **Μέγεθος Αρμονικής Μνήμης (Harmony Memory Size – HMS)**
  Είναι το πλήθος των αρμονιών που αποθηκεύονται στη μνήμη του υπολογιστή. Αν μια καινούργια αρμονία είναι καλή τότε αντικαθιστά κάποιαν άλλη ώστε να μη ξεπεραστεί το μέγεθος της μνήμης.

- **Δείκτη Αποδοχής της Αρμονικής Μνήμης (Harmony Memory Consideration Rate)**
  Όταν δημιουργείται μια νέα αρμονία υπάρχει πιθανότητα να λαμβάνονται υπόψη αρμονίες που ήδη υπάρχουν στην Αρμονική Μνήμη. Αυτό το ποσοστό ονομάζεται Δείκτης Αποδοχής της Αρμονικής Μνήμης και συνήθως λαμβάνει μεγάλες τιμές(άνω του 50%).

- **Δείκτης Προσαρμογής Τόνου (Pitch Adjustment Rate)**
  Όταν λαμβάνονται υπόψη υπάρχουσες αρμονίες, στη δημιουργία μιας νέας, τότε υπάρχει περίπτωση η νέα αυτή αρμονία να υποστεί μια ελαφρά παράλλαξη. Η πιθανότητα να συμβεί αυτό ονομάζεται Δείκτης Προσαρμογής Τόνου και συνήθως λαμβάνει μικρές τιμές(κάτω από 50%).

  $\mathbf{X_i^{new} = X_i + rand()\cdot bw}$
  bw: *το εύρος που μπορεί να αποκλίνει η παραλλαγή*
  rand(): *τυχαία τιμή από το 0 έως το 1*

  Ο συγκεκριμένος μηχανισμός είναι πολύ σημαντικός διότι εξασφαλίζει ότι η αναζήτηση δεν εγκλωβίζεται σε τοπικά μέγιστα.

Δομή του Αλγορίθμου

Η δομή του αλγορίθμου μπορεί να αναλυθεί σε 5 βασικά βήματα.

I) <u>Καθορισμός του Προβλήματος και Ρύθμιση των Παραμέτρων του Αλγορίθμου</u>

Αρχικά πρέπει να οριστεί από το πρόβλημα αν ο αλγόριθμος θα αναζητήσει την μεγιστοποίηση ή την ελαχιστοποίηση της αντικειμενικής συνάρτησης.

$$max(or\ min)F(x)$$

Όπου F είναι η αντικειμενική συνάρτηση που πρόκειται να βελτιστοποιηθεί και X είναι ένα διάνυσμα που αποτελείται από δύο ή περισσότερες μεταβλητές. Αυτές οι μεταβλητές είναι οι μεταβλητές απόφασης οι οποίες καθορίζουν την τιμή που θα παραχθεί από την αντικειμενική συνάρτηση. Κατόπιν, ορίζεται(από το πρόβλημα) το πεδίο ορισμού των μεταβλητών απόφασης. Αυτό σημαίνει ότι για κάθε μεταβλητή απόφασης πρέπει να ισχύει:

$$_Lx_i < X_i < {}_Ux_i$$

όπου ${}_{L}x_i$ το κατώτατο όριο του πεδίου ορισμού και ${}_{U}x_i$ το ανώτατο όριο του πεδίου ορισμού. Παράλληλα, θα πρέπει να ρυθμιστεί κατάλληλα ο αλγόριθμος ώστε να είναι όσο πιο αποδοτικός γίνεται. Αυτές οι ρυθμίσεις αφορούν το **Μέγεθος** της **Αρμονικής Μνήμης(Harmony Memory Size - HMS),** τον **Δείκτη Αποδοχής** της **Αρμονικής Μνήμης(Harmony Memory Consideration Rate - HMCR)**, και του **Δείκτη Προσαρμογής** του **Τόνου(Pitch Adjustment Rate – PAR).**

II) Δημιουργία Μνήμης Αρμονιών

Δημιουργείται ένα σύνολο διανυσμάτων που θα αποτελέσουν την Αρμονική Μνήμη(HM). Η κάθε μεταβλητή απόφασης του κάθε διανύσματος λαμβάνει μια τυχαία τιμή τηρώντας φυσικά τους περιορισμούς. Κατόπιν, ταξινομείται η μνήμη με βάση την αντικειμενική συνάρτηση.

III) Δημιουργία Νέας Αρμονίας

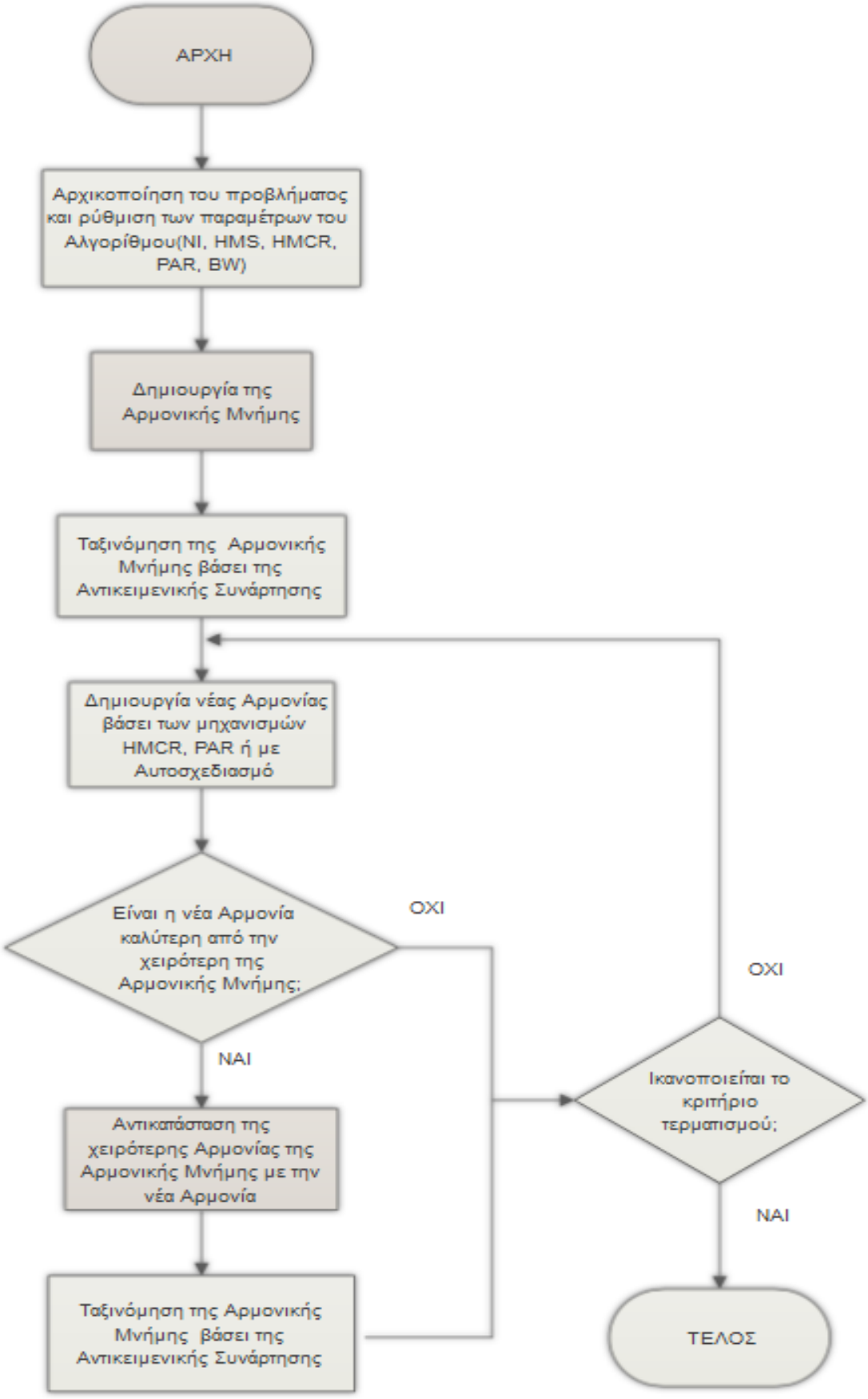
Η νέα αρμονία δημιουργείται με βάση τον **Δείκτη Αποδοχής Αρμονικής Μνήμης(HMCR)** και με βάση τον **Δείκτη Προσαρμογής Τόνου(PAR)**. Η πιθανότητα να χρησιμοποιηθούν αυτοί οι δύο μηχανισμοί(HMCR, PAR) είναι ανάλογη της τιμής που έλαβαν κατά την ρύθμιση του Αλγορίθμου(Βήμα Ι). Διαφορετικά, η νέα αρμονία δημιουργείται με την παραγωγή ενός εντελώς καινούργιου διανύσματος που λαμβάνει εντελώς τυχαίες τιμές(τηρώντας φυσικά τους περιορισμούς). Σ' αυτή την περίπτωση δεν λαμβάνεται καθόλου υπόψη η Αρμονική Μνήμη και η πιθανότητα να συμβεί αυτό είναι **1-HMCR**.

IV) Ενημέρωση της Αρμονικής Μνήμης

Η νέα αρμονία που δημιουργήθηκε στο προηγούμενο βήμα πρέπει τώρα να αξιολογηθεί. Εάν έχει καλύτερη τιμή(με βάση την αντικειμενική συνάρτηση) από την χειρότερη αρμονία της Μνήμης τότε την αντικαθιστά. Σε αυτή την περίπτωση η Αρμονική Μνήμη θα πρέπει να ταξινομηθεί εκ νέου. Αν η νέα αρμονία είναι χειρότερη από την χειρότερη αρμονία της Μνήμης τότε η νέα αρμονία διαγράφεται και κατά συνέπεια η Αρμονική Μνήμη παραμένει ίδια.

V) Επανάληψη

Αφού ολοκληρώθηκαν όλα τα παραπάνω βήματα τότε επαναλαμβάνονται τα βήματα ΙΙ, ΙΙΙ και IV. Αυτά τα βήματα θα συνεχίσουν να επαναλαμβάνονται μέχρις ότου ο αριθμός των επαναλήψεων εξαντληθεί(το πλήθος των επαναλήψεων ορίζεται στο Βήμα Ι) ή μέχρι να τηρηθεί κάποιο άλλο κριτήριο τερματισμού.


ΑΡΧΗ
Αρχικοποίηση του προβλήματος και ρύθμιση των παραμέτρων του Αλγορίθμου(NI, HMS, HMCR, PAR, BW)
Δημιουργία της Αρμονικής Μνήμης
Ταξινόμηση της Αρμονικής Μνήμης βάσει της Αντικειμενικής Συνάρτησης
Δημιουργία νέας Αρμονίας βάσει των μηχανισμών HMCR, PAR ή με Αυτοσχεδιασμό
Είναι η νέα Αρμονία καλύτερη από την χειρότερη της Αρμονικής Μνήμης;
ΟΧΙ
ΝΑΙ
Αντικατάσταση της χειρότερης Αρμονίας της Αρμονικής Μνήμης με την νέα Αρμονία
Ταξινόμηση της Αρμονικής Μνήμης βάσει της Αντικειμενικής Συνάρτησης
Ικανοποιείται το κριτήριο τερματισμού;
ΟΧΙ
ΝΑΙ
ΤΕΛΟΣ

Δημιουργία Αρμονίας - Λύσης

Στον αλγόριθμο harmony search η σημαντικότερη, ίσως, διαδικασία είναι η διαδικασία του αυτοσχεδιασμού νέας αρμονίας. Η δημιουργία δηλαδή μιας νέας υποψήφιας λύσης του προβλήματος. Για να πάρει μια νότα(μεταβλητή απόφασης) τιμή λαμβάνονται υπόψιν τρεις μηχανισμοί. Οι μηχανισμοί αυτοί είναι ο HMCR, ο PAR και η παραγωγή τυχαίας τιμής. Αναλυτικότερα:

I) Αρχικά δημιουργείται ένα διάνυσμα(αρμονία) το οποίο είναι άδειο. Ένα διάνυσμα το οποίο δεν έχει καμία απολύτως τιμή(null). Φυσικά, το διάνυσμα θα πρέπει να έχει τις διαστάσεις και το είδος της τιμής(ακέραια ή πραγματική) που ορίστηκαν νωρίτερα στο πρόγραμμα. Αφού αρχικοποιηθεί το νέο διάνυσμα τότε εκτελείται ένας βρόγχος όπου επαναλαμβάνονται τα παρακάτω βήματα. Ο αριθμός των επαναλήψεων είναι ίσος με τον αριθμό των διαστάσεων του διανύσματος.

II) Παράγεται μια τυχαία δεκαδική τιμή από το 0 έως το 1. Αν η τυχαία τιμή που παράχθηκε είναι μεγαλύτερη από την τιμή HMCR τότε ο αλγόριθμος συνεχίζει από το βήμα IV αγνοώντας τα ενδιάμεσα βήματα. Διαφορετικά, αν η τυχαία τιμή είναι μικρότερη από την τιμή HMCR, τότε παράγεται εκ νέου μια νέα τυχαία τιμή ανάμεσα στο 0 και την τιμή που αντιστοιχεί στο μέγεθος της Μνήμης(HMS). Από την τιμή που παράχθηκε επιλέγεται το αντίστοιχο διάνυσμα της Αρμονικής Μνήμης(Harmony Memory). Στη συνέχεια, καταχωρείται στη i-στή μεταβλητή απόφασης του νέου διανύσματος, η i-στή μεταβλητή απόφασης του διανύσματος που επιλέχθηκε νωρίτερα.

III) Εφόσον, πραγματοποιήθηκε η μέθοδος HMCR τότε παράγεται εκ νέου μια νέα τυχαία τιμή από το 0 έως το 1 όπως προηγουμένως. Αν αυτή η τυχαία τιμή είναι μεγαλύτερη από την τιμή PAR τότε αυτό το βήμα αγνοείται. Αν είναι μικρότερη από την τιμή PAR τότε θα πρέπει να μεταβληθεί ελαφρώς η i-στή μεταβλητή του διανύσματος. Αυτό σημαίνει ότι θα προσθέσουμε στην i-στή μεταβλητή απόφασης μια τιμή ίση με το εύρος του τόνου(σταθερά που ορίζεται στην αρχή του προγράμματος).

IV) Σε περίπτωση που το βήμα II δεν υλοποιήθηκε(κατ' επέκταση και το βήμα III) τότε η i-στή μεταβλητή απόφασης λαμβάνει μια εντελώς τυχαία τιμή που πρέπει όμως να είναι εντός των ορίων του πεδίου ορισμού.

VI. ΠΑΡΑΛΛΑΓΕΣ ΤΟΥ ΑΛΓΟΡΙΘΜΟΥ HARMONY SEARCH

Επισκόπηση

Με την πάροδο του χρόνου, ο αλγόριθμος Harmony Search χρησιμοποιείται σε όλο και περισσότερες εφαρμογές. Πολλές φορές όμως, και ανάλογα με την φύση του εκάστοτε προβλήματος γίνονται τροποποιήσεις σε σχέση με τον κλασσικό αλγόριθμο Harmony Search. Οι τροποποιήσεις αυτές αποσκοπούν στην επίτευξη ταχύτερης σύγκλισης στο βέλτιστο και συνολικά καλύτερης απόδοσης. Το αντικείμενο αυτών των παραλλαγών είναι συνήθως οι παράμετροι της μεθόδου ή η τροποποίηση κάποιου από τους επιμέρους μηχανισμούς. Η ρύθμιση των παραμέτρων της μεθόδου, είναι ζωτικής σημασίας και σε μεγάλο βαθμό επηρεάζει την απόδοση που αυτή θα έχει. Σε αυτό το κεφάλαιο θα παρουσιαστούν οι σημαντικότερες παραλλαγές του αλγορίθμου HS. Συχνά οι τροποποιήσεις που γίνονται συνδέονται άμεσα με το πρόβλημα που πρόκειται να αντιμετωπιστεί, συνεπώς η παρουσίαση όλων των παραλλαγών που έχουν γίνει στον αλγόριθμο HS είναι εκ των πραγμάτων αδύνατη.

Μέθοδος	**Αγγλική Ονομασία**	**Έτος**	**Δημιουργός**
Βελτιωμένη Αναζήτηση Αρμονίας	Improved Harmony Search	2007	M.Mahdavi, M.Fesanghary, E.Damangir
Συνολικά Βέλτιστη Αναζήτηση Αρμονίας	Global Best Harmony Search	2008	Mohamed Omran, Mehrdad Mahdavi
Αυτο-Προσαρμοστική Αναζήτηση Αρμονίας	Self Adaptive Harmony Search	2009	Chia-Ming Wang, Yin-Fu Huang
Ανεξάρτητη από Παραμέτρους Αναζήτηση Αρμονίας	Parameter Setting-Free Harmony Search	2010	Zong Woo Geem, Kwee Bo Sim
Αναζήτηση Μελωδίας	Melody Search	2012	S.M. Ashrafi, A.B.Dariane

Κλασσική Αναζήτηση Αρμονίας υπό την Επίδραση του Συνόλου των Μουσικών

Ο κλασσικός αλγόριθμος Harmony Search χρησιμοποιεί 3 μηχανισμούς για να δημιουργήσει μια νέα αρμονία. Το 2006 ο Zong Woo Geem πρότεινε την προσθήκη ενός νέου μηχανισμού μέσω του οποίου θα λαμβάνονται υπόψιν τυχόν συσχετίσεις που μπορεί να υπάρχουν μεταξύ διαφορετικών μεταβλητών απόφασης. Η έμπνευση αυτού του μηχανισμού προήλθε και πάλι από την μουσική. Σε ένα μουσικό σχήμα πολλές φορές κάποια μουσικά όργανα έχουν πολύ στενή σχέση. Η μίμηση των τόνων που παίζει ένα όργανο από ένα άλλο, οι συνεχείς ερωτήσεις – απαντήσεις, ο μουσικός διάλογος είναι συνηθισμένα φαινόμενα στα περισσότερα είδη μουσικής. Για παράδειγμα, σε μια συμφωνική ορχήστρα τα πρώτα βιολιά βρίσκονται σε διαρκή διάλογο με τα δεύτερα βιολιά τα οποία συνοδεύουν και απαντούν. Είναι προφανές ότι αυτές οι δύο ομάδες βιολιών έχουν πολύ στενότερη σχέση μεταξύ τους σε σύγκριση με τα τύμπανα ή τα πνευστά όργανα. Κατ' αναλογία, σε πολλά προβλήματα βελτιστοποίησης η αντικειμενική συνάρτηση ενδεχομένως να περιέχει μεταβλητές απόφασης που δεν είναι ανεξάρτητες μεταξύ τους. Κάθε νέα μεταβλητή απόφασης μιας νέας αρμονίας μπορεί να βασιστεί στην τιμή μιας άλλης μεταβλητής της ίδιας αρμονίας εφόσον διαπιστωθεί ότι έχουν στενή σχέση.

$$\mathbf{x_i = max[Corr(x_i, x_j)]^2}$$

Corr(): *Συνάρτηση στατιστικής συσχέτισης 2 μεταβλητών*
max()*: Μέγιστη συσχέτιση μεταξύ 2 μεταβλητών*

$$\mathbf{x_i^{new} = f_n(x_j^{new})}$$

f_n(): *Συνάρτηση μέσω της οποίας παράγεται μια τιμή που θα είναι βασισμένη στην παράμετρο που θα πάρει*

Για να υλοποιηθεί αυτός ο μηχανισμός ορίζεται μια νέα παράμετρος που λέγεται **ECR(Ensemble Consideration Rate)** και λαμβάνει τιμές που ανήκουν στο διάστημα (0,1). Η τιμή ECR αντικατοπτρίζει την πιθανότητα να δημιουργηθεί μια νέα μεταβλητή απόφασης με βάση τον μηχανισμό της επίδρασης του συνόλου των μουσικών.

Βελτιωμένη Αναζήτηση Αρμονίας

Η **Βελτιωμένη Αναζήτηση Αρμονίας(Improved Harmony Search)** προτάθηκε για πρώτη φορά από τους M.Mahdavi, M.Fesanghary και E.Damangir το 2007. Οι παράμετροι HMCR και PAR που αναφέρθηκαν και παραπάνω βοηθούν τον αλγόριθμο να αναζητήσει καλές λύσεις συνολικά(HMCR) και τοπικά(PAR). Ο κλασσικός αλγόριθμος Harmony Search χρησιμοποιεί σταθερές τιμές για αυτές τις παραμέτρους όπως επίσης και για την παράμετρο BW. Σε αυτή την παραλλαγή του αλγορίθμου προτείνεται μια διαφορετική προσέγγιση. Συγκεκριμένα, προτείνεται να μένει σταθερή μόνο η τιμή HMCR και να αλλάζουν δυναμικά οι παράμετροι PAR και BW. Όσο το πρόγραμμα εκτελείται τόσο περισσότερο αυξάνεται η τιμή PAR.

$$\mathbf{PAR_{(gn)} = PAR_{min} + [(PAR_{max} - PAR_{min})/NI] * gn}$$

PAR: *Δείκτης προσαρμογής τόνου*
PAR_{min}: *Ελάχιστη τιμή του PAR*
PAR_{max}: *Μέγιστη τιμή του PAR*
NI: *Μέγιστος αριθμός επαναλήψεων*
gn: *Τρέχων αριθμός επανάληψης*

Αντίθετα η τιμή BW ακολουθεί την αντίθετη διαδρομή. Όσο περισσότερο προχωράει η αναζήτηση τόσο μειώνεται η τιμή της.

$$\mathbf{BW(gn) = Bw_{max} * exp(c*gn)},$$
$$\mathbf{c = ln(Bw_{max}/Bw_{min})/NI}$$

BW: *Εύρος προσαρμογής του PAR σε κάθε επανάληψη*
BW_{min}: *Ελάχιστη τιμή του BW*
BW_{max}: *Μέγιστη τιμή του BW*

Συνολικά Βέλτιστη Αναζήτηση Αρμονίας

Η **Συνολικά Βέλτιστη Αναζήτηση Αρμονίας(Global Best Harmony Search)** είναι μια ακόμα παραλλαγή του κλασσικού αλγορίθμου Harmony Search που προτάθηκε το 2008 από τον Mohamed Omran και τον Mehrdad Mahdavi. Η συγκεκριμένη παραλλαγή άντλησε έμπνευση από έναν άλλο μεταευρετικό αλγόριθμο, την Βελτιστοποίηση Σμήνους Σωματιδίων(Particle Swarm Intelligence). Σε αυτή την προσέγγιση και πάλι τροποποιείται η μέθοδος Προσαρμογής του Τόνου(PAR). Συγκεκριμένα, για κάθε νέα νότα(μεταβλητή απόφασης) της κάθε νέας αρμονίας που δημιουργείται, λαμβάνονται υπόψιν οι νότες τις καλύτερης αρμονίας που βρίσκεται εκείνη τη στιγμή στη μνήμη. Αυτό επιτυγχάνεται καταχωρώντας στην μεταβλητή που εξετάζεται μια τυχαία μεταβλητή από την καλύτερη αρμονία της μνήμης. Ενώ παράλληλα, η παράμετρος BW δεν χρησιμοποιείται καθόλου. Ωστόσο, η επιλογή μιας άλλης μεταβλητής(από αυτήν που εξετάζεται), χωρίς να υπάρχει κάποιος έλεγχος συσχέτισης, είναι παράδοξη. Τα αποτελέσματα όμως στα προβλήματα δοκιμών δείχνουν ότι η συγκεκριμένη παραλλαγή είναι αρκετά αποδοτική. Συχνά εντοπίζει τη βέλτιστη λύση γρηγορότερα από τις υπόλοιπες παραλλαγές όμως έχει το μειονέκτημα ότι μερικές φορές συγκλίνει πρόωρα και παγιδεύεται σε τοπικά ακρότατα. Όταν η παραλλαγή GB-HS πρόκειται να εφαρμοστεί σε προβλήματα βελτιστοποίησης της πραγματικής ζωής τότε δεν παρουσιάζει την ίδια απόδοση που παρουσιάζει στις μαθηματικές συναρτήσεις δοκιμών(benchmarks). Αυτό συμβαίνει κυρίως διότι στα πραγματικά προβλήματα κάποιες μεταβλητές σχετίζονται μεταξύ τους, σε αντίθεση με τα benchmarks.

Αυτο-Προσαρμοστική Αναζήτηση Αρμονίας

Η **Αυτο-Προσαρμοστική Αναζήτηση Αρμονίας(Self Adaptive Harmony Search)** προτάθηκε το 2009 από τους Chia-Ming Wang και Yin-Fu Huang. Όπως οι προηγούμενες παραλλαγές έτσι και αυτή προτείνει τροποποιήσεις στον μηχανισμό Προσαρμογής Τόνου(PAR). Συγκεκριμένα, προτείνεται η γραμμική μείωση του συντελεστή PAR από 1 σε 0. Ταυτόχρονα, συμμετέχουν η μέγιστη και η ελάχιστη μεταβλητή της μνήμης στη δημιουργία νέας μεταβλητής ορίζοντας το κατώτατο και το ανώτατο όριο της νέας τιμής. Η παράμετρος BW δεν χρησιμοποιείται καθόλου.

$$\mathbf{x_i + [max(HM_i) - x_i]*random(0,1)}, \text{ ή}$$
$$\mathbf{x_i - [x_i - min(HM_i)]*random(0,1)}$$

***x_i**: Μεταβλητή της νέας Αρμονίας*
***max(HM_i)**: Μέγιστη τιμή της μεταβλητής στη μνήμη*
***min(HM_i)**: Μέγιστη τιμή της μεταβλητής στη μνήμη*
***random(0,1)**: Τυχαία τιμή από 0 έως 1*

Με αυτό τον τρόπο, κάθε νέα αρμονία μπορεί να αξιοποιήσει καλύτερα την εμπειρία που υπάρχει μέχρι εκείνη την στιγμή στη μνήμη.

Ανεξάρτητη από Παραμέτρους Αναζήτηση Αρμονίας

Η **Ανεξάρτητη από Παραμέτρους Αναζήτηση Αρμονίας**(**Parameter Setting free Harmony Search**) διαφέρει από τις υπόλοιπες παραλλαγές του αλγορίθμου Harmony Search υπό την έννοια ότι δεν απαιτείται ρύθμιση παραμέτρων. Προτάθηκε το 2010 από τον Zong Woo Geem και τον Kwee Bo Sim. Σε αυτή την παραλλαγή ο αλγόριθμος αρχικά εκτελείται λίγες φορές κατά τον κλασσικό τρόπο. Κατόπιν, δημιουργείται ένας επιπλέον πίνακας που θα έχει τον ίδιο αριθμό στοιχείων που έχει και η Αρμονική Μνήμη. Ο πίνακας αυτός θα ονομάζεται Μνήμη Λειτουργικού Τύπου και θα περιέχει την προέλευση των αντίστοιχων στοιχείων της Αρμονικής Μνήμης. Οπότε, οι παράμετροι HMCR και PAR λαμβάνουν τιμή που εξαρτάται από το πλήθος των στοιχείων που προήλθαν από τον εκάστοτε μηχανισμό.

HMCR = Πλήθος στοιχείων που προήλθαν μέσω HMCR ή PAR/HMS
PAR = ***Πλήθος στοιχείων που προήλθαν μέσω PAR/HMS***

Με αυτό τον τρόπο οι παράμετροι HMCR και PAR μεταβάλλονται συνεχώς κατά τη διάρκεια εκτέλεσης του προγράμματος χωρίς να χρειάζεται να ρυθμιστούν ρητά.

Είναι εύκολο να διαπιστωθεί πως αν η παράμετρος HMCR λάβει την τιμή 1 τότε δεν θα αλλάξει ξανά κατά τη διάρκεια της αναζήτησης. Το ίδιο συμβαίνει και με την παράμετρο PAR όταν λαμβάνει την τιμή 0. Για αυτό τον λόγο εφαρμόζεται ένας ακόμα τύπος προκειμένου να αποφευχθεί πρόωρη σύγκλιση σε κάποιο τοπικό ακρότατο.

$$HMCR = HMCR + \Phi_{HMCR} * random(0,1)$$

Αν η παραπάνω παράσταση δεν ανήκει στο διάστημα (0,1) τότε το HMCR παραμένει ίδιο. Όπου Φ_{HMCR} *= Συντελεστής θορύβου HMCR*

$$PAR = PAR + \Phi_{PAR} * random(0,1)$$

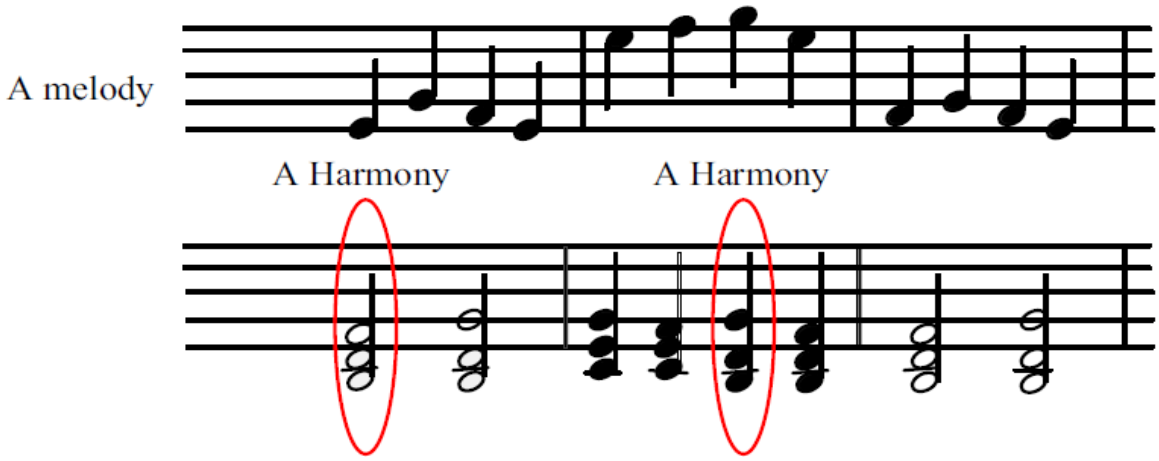
Αν η παραπάνω παράσταση δεν ανήκει στο διάστημα (0,1) τότε το PAR παραμένει ίδιο. Όπου Φ_{HPAR} *= Συντελεστής θορύβου PAR*

Αναζήτηση Μελωδίας

Η **Αναζήτηση Μελωδίας(Melody Search)** διαφέρει σημαντικά από τον κλασσικό αλγόριθμο Harmony Search και κατά συνέπεια και από τις παραλλαγές που παρουσιάστηκαν στις προηγούμενες σελίδες. Ωστόσο, έχουν κοινή έμπνευση την μουσική και ο τρόπος αυτοσχεδιασμού μιας νέας υποψήφιας λύσης(μελωδίας) ομοιάζει αρκετά με τους αντίστοιχους μηχανισμούς του αλγορίθμου Harmony Search. Παρουσιάστηκε για πρώτη φορά το 2012 από τους S.M. Ashrafi και A.B.Dariane.

Όπως αναφέρθηκε και προηγουμένως, στην Μουσική, αρμονία ονομάζεται η ταυτόχρονη συνήχηση δύο ή περισσότερων φθόγγων. Ως μελωδία χαρακτηρίζεται η γραμμική αλληλουχία ξεχωριστών τόνων/ήχων. Η αρμονία είναι η κάθετη διαφορά δύο τονικών υψών(διάστημα), ενώ η μελωδία είναι η οριζόντια διαφορά δύο τονικών υψών.

Όπως στην μέθοδο Harmony Search έτσι και στη μέθοδο Melody Search, ο αλγόριθμος μιμείται την διαδικασία που ακολουθεί ένα μουσικό σχήμα μέχρι να βρει την βέλτιστη μελωδία για ένα μουσικό κομμάτι. Η κάθε νότα αντιστοιχεί σε μια μεταβλητή απόφασης και η κάθε μελωδία σε μια υποψήφια λύση. Παρόλο που ο αλγόριθμος Melody Search(MS) υιοθετεί τις βασικές αρχές της μεθόδου Harmony Search(HS) η δομή τους είναι αρκετά διαφορετική. Στον MS χρησιμοποιούνται αρκετά διανύσματα που αντιπροσωπεύουν τη μνήμη του κάθε παίκτη ξεχωριστά, εν αντιθέσει με τον HS που χρησιμοποιεί μια μόνο μνήμη για ολόκληρο το μουσικό σχήμα. Επιπλέον, στον MS αλγόριθμο το πεδίο ορισμού ανανεώνεται σε κάθε επανάληψη. Ο βασικός κορμός του αλγορίθμου χωρίζεται σε δύο φάσεις. Αρχικά, ο κάθε παίκτης αυτοσχεδιάζει τη δική του μελωδία ξεχωριστά. Οι μελωδίες που δημιουργούνται χρησιμοποιούν σταθερό εύρος



Ashrafi, S.M., Dariane, A.B., "Performance evaluation of an improved harmony seach algorithm for numerical optimization: Melody Search(MS)", (2012), pp. 1301-1321

τόνου(πεδίο ορισμού). Σε δεύτερη φάση, το συγκρότημα ως σύνολο αυτοσχεδιάζει μελωδίες. Η ύπαρξη διαφορετικών μελωδιών μπορεί να οδηγήσει ένα συγκρότημα στην

επιλογή καλύτερων τόνων και κατά συνέπεια καλύτερων μελωδιών. Σε κάθε επανάληψη το εύρος τόνου ενημερώνεται.

Σε μορφή ψευδοκώδικα ο αλγόριθμος γράφεται όπως φαίνεται παρακάτω:

```
Αρχικοποίησε την μνήμη του κάθε μουσικού.
While (κριτήριο επαναλήψεων) Do
     Αυτοσχεδίασε μια νέα μελωδία από κάθε PM(Player Memory).
     Ενημέρωσε κάθε PM.
End While
While (κριτήριο επαναλήψεων) Do
     Αυτοσχεδίασε μια νέα μελωδία από κάθε PM μέσα στο
     επιτρεπτό εύρος τόνου.
     Ενημέρωσε κάθε PM.
     Προσδιόρισε το νέο εύρος τόνου
End While
```

VII. ΠΡΟΒΛΗΜΑΤΑ ΔΟΚΙΜΩΝ

Επισκόπηση

Τα **προβλήματα δοκιμών(benchmarks)** χρησιμοποιούνται για να εξαχθούν συμπεράσματα και να αξιολογηθούν κάποια χαρακτηριστικά των αλγορίθμων βελτιστοποίησης. Τα χαρακτηριστικά που εξετάζονται είναι ο **ρυθμός σύγκλισης**, η **ακρίβεια** και η **ικανότητα του αλγορίθμου να παραμένει αποτελεσματικός σε διαφορετικές συνθήκες**. Πέραν όμως της αποτελεσματικότητας και της απόδοσης ενός αλγορίθμου, τα benchmarks δίνουν τη δυνατότητα στους προγραμματιστές να κατανοήσουν σε μεγαλύτερο βάθος τις ανάγκες ρύθμισης των παραμέτρων του αλγορίθμου. Η συγκεκριμένη πρακτική προτείνεται για την αξιολόγηση κάθε Μεταευρετικού αλγορίθμου.

Τα περισσότερα benchmarks αποτελούνται από μια μαθηματική σχέση όπου η κάθε μεταβλητή απόφασης έχει συγκεκριμένο πεδίο ορισμού και το βέλτιστο αποτέλεσμα είναι γνωστό. Το βέλτιστο αποτέλεσμα είναι η μέγιστη ή η ελάχιστη τιμή που μπορεί να παράξει αυτή η σχέση. Όταν ο αλγόριθμος τρέχει, το ζητούμενο είναι πόσο γρήγορα βρίσκει την βέλτιστη λύση και με πόση ακρίβεια.

Στη συνέχεια αυτού του κεφαλαίου θα δοκιμαστεί ο αλγόριθμος Harmony Search και θα αξιολογηθεί η απόδοση που έχει στο εκάστοτε benchmark. Η παραλλαγή που χρησιμοποιείται συνδυάζει την κλασσική Αναζήτηση Αρμονίας και την βελτιωμένη Αναζήτηση Αρμονίας. Συγκεκριμένα, χρησιμοποιείται σταθερός συντελεστής PAR αλλά ο συντελεστής BW μειώνεται σταδιακά με το πέρασμα του χρόνου. Οι υπόλοιποι μηχανισμοί παραμένουν ίδιοι όπως περιγράφηκαν στα προηγούμενα κεφάλαια. Αξίζει να αναφερθεί, ότι η τιμή της αντικειμενικής συνάρτησης για κάθε εκτέλεση επιλέγεται δειγματοληπτικά ως ο μέσος όρος 10 εκτελέσεων με τις παραμέτρους να λαμβάνουν τις τιμές που αναγράφονται κάθε φορά.

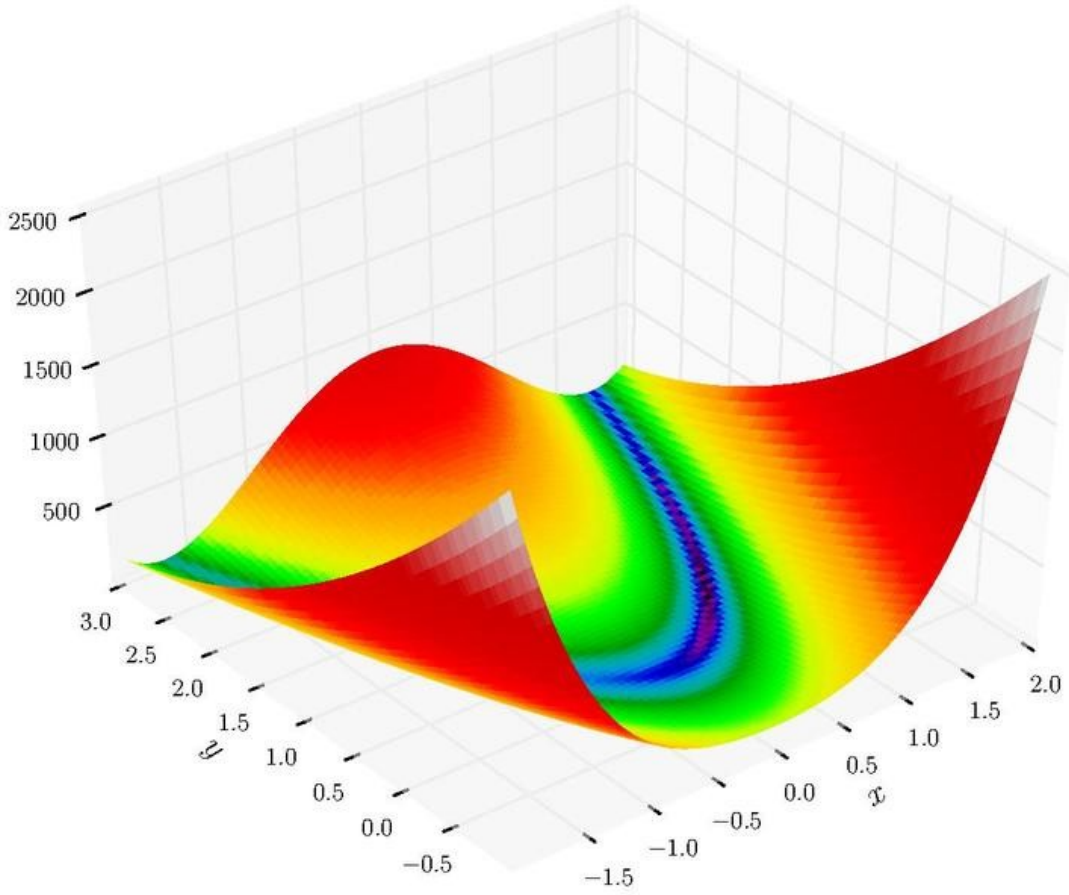
Η συνάρτηση Rosenbrock

Η συνάρτηση Rosenbrock προτάθηκε το 1960 από τον Howard H. Rosenbrock. Είναι γνωστή

ως “η κοιλάδα του Rosenbrock” ή και “η συνάρτηση μπανάνα του Rosenbrock”. Η συνάρτηση έχει την ακόλουθη μορφή:

$$f(\mathbf{x}) = \sum_{i=1}^{d-1} \left[100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2\right]$$

Το ολικό ελάχιστο της συνάρτησης είναι **f(x) = 0** για $\mathbf{x_i = 1}$ με πεδίο ορισμού
$\mathbf{-2.048 \leq x_i \leq 2.048}$



https://upload.wikimedia.org/wikipedia/commons/thumb/3/32/Rosenbrock_function.svg/300px-Rosenbrock_function.svg.png

Σε προβλήματα δύο διαστάσεων, το ολικό βέλτιστο εντοπίζεται μέσα σε μια μακριά, στενή και παραβολικής μορφής κοιλάδα. Η συνάρτηση μπορεί να επεκταθεί και για προβλήματα άνω των 2 διαστάσεων(μεταβλητών απόφασης) όπου φυσικά αυξάνεται και ο βαθμός δυσκολίας.

Ο αλγόριθμος δοκιμάστηκε στη συνάρτηση Rosenbrock δύο διαστάσεων. Στις δοκιμές, χρησιμοποιήθηκαν διαφορετικοί συνδυασμοί παραμέτρων για μελετηθεί καλύτερα η συμπεριφορά του αλγορίθμου. Σε όλους τους συνδυασμούς παραμέτρων χρησιμοποιήθηκαν ίδιες τιμές για τον συντελεστή BW. Συγκεκριμένα, ο συντελεστής BW έλαβε τιμές που

κυμαίνονται από 0.5 έως 0.1, καθώς οι τιμές του μειώνονται γραμμικά με το πέρασμα των επαναλήψεων.

NI	HMS	HMCR	PAR	ΜΕΣΟΣ ΟΡΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ(10 ΕΚΤΕΛΕΣΕΙΣ)
500	10	0.6	0.1	0.085246
			0.5	0.060969
		0.9	0.1	0.296348
			0.5	0.075068
	30	0.6	0.1	0.060309
			0.5	0.104357
		0.9	0.1	0.131963
			0.5	0.135233
	50	0.6	0.1	0.08112
			0.5	0.075469
		0.9	0.1	0.101136
			0.5	0.072295
2000	10	0.6	0.1	0.047224
			0.5	0.024111
		0.9	0.1	0.135196
			0.5	0.006847
	30	0.6	0.1	0.029393
			0.5	0.030479
		0.9	0.1	0.123368
			0.5	0.029464
	50	0.6	0.1	0.036309

			0.5	0.025317
		0.9	0.1	0.051372
			0.5	0.031175
5000	10	0.6	0.1	0.015437
			0.5	0.001944
		0.9	0.1	0.053132
			0.5	0.000173
	30	0.6	0.1	0.021295
			0.5	0.00428
		0.9	0.1	0.049756
			0.5	0.001046
	50	0.6	0.1	0.007344
			0.5	0.005927
		0.9	0.1	0.039993
			0.5	0.003148

Στον παραπάνω πίνακα φαίνονται όλες οι εκτελέσεις του αλγορίθμου και οι ρυθμίσεις που είχε κάθε φορά. Με πράσινο χρώμα σκιαγραφείται ο καλύτερος συνδυασμός παραμέτρων για κάθε πλήθος επαναλήψεων.

Συμπερασματικά, μπορεί να παρατηρηθεί ότι σε όλες τις περιπτώσεις η καλύτερη τιμή παράγεται όταν η Αρμονική Μνήμη(HMS) έχει μικρή χωρητικότητα. Παράλληλα, ο συντελεστής PAR φαίνεται να έχει θετική επίδραση στην αναζήτηση όταν λαμβάνει μεγάλες τιμές. Όσον αφορά τον συντελεστή HMCR παρατηρείται ότι για υψηλές τιμές αποδίδει μόνο όταν οι επαναλήψεις είναι πολλές.

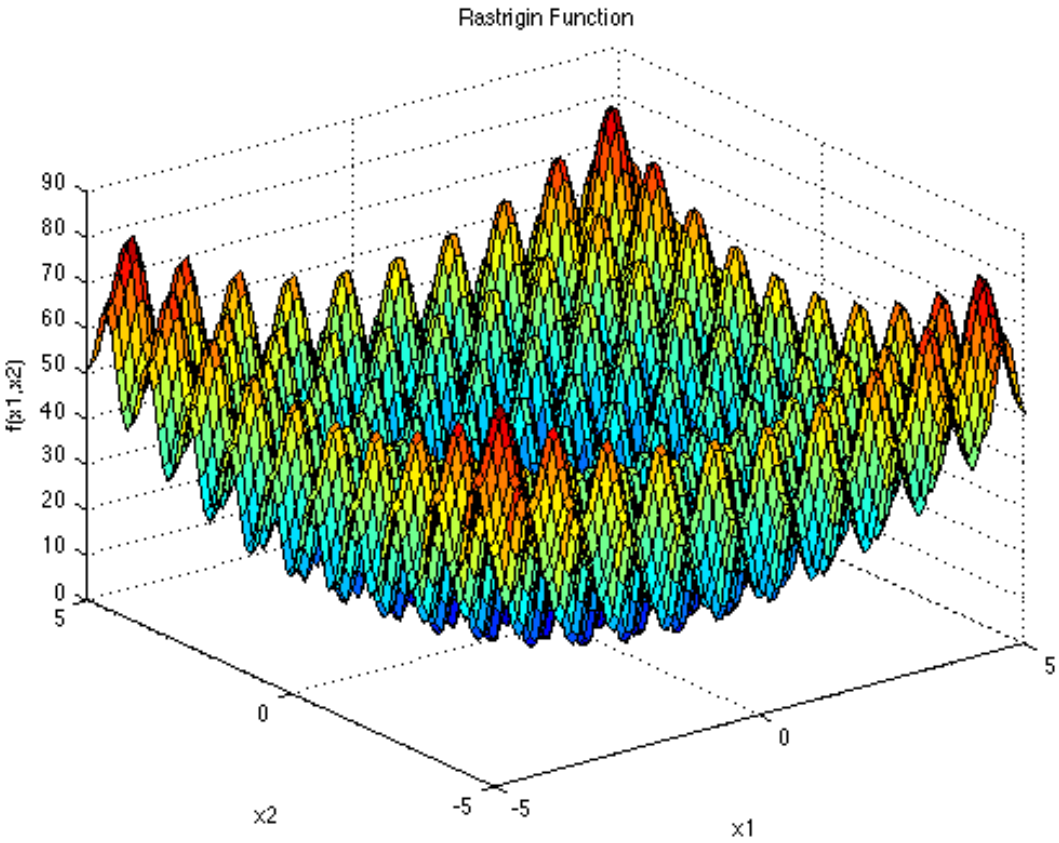
Η συνάρτηση Rastrigin

Η συνάρτηση Rastrigin προτάθηκε αρχικά από τον Rastrigin και αργότερα επεκτάθηκε και γενικεύτηκε από τον Heinz Mühlenbein. Θεωρείται δύσκολη συνάρτηση λόγω του μεγάλου πεδίου ορισμού αλλά και επειδή έχει πολλά τοπικά ελάχιστα. Η ύπαρξη πολλών τοπικών ακρότατων είναι μια δύσκολη πρόκληση για κάθε μεταευρετικό αλγόριθμο διότι πολλές φορές αποπροσανατολίζονται και η αναζήτηση οδηγείται προς λάθος κατεύθυνση.

Η συνάρτηση Rastrigin ορίζεται ως εξής:

$$f(\mathbf{x}) = 10d + \sum_{i=1}^{d} \left[x_i^2 - 10\cos(2\pi x_i) \right]$$

Το ολικό ελάχιστο της συνάρτησης είναι **f(x) = 0** για $\mathbf{x_i = 0}$ με πεδίο ορισμού $\mathbf{-5.12 \leq x_i \leq 5.12}$



https://www.sfu.ca/~ssurjano/rastr.png

Ο αλγόριθμος δοκιμάστηκε στη συνάρτηση Rosenbrock δύο διαστάσεων. Στις δοκιμές, χρησιμοποιήθηκαν διαφορετικοί συνδυασμοί παραμέτρων για μελετηθεί καλύτερα η

συμπεριφορά του αλγορίθμου. Όπως και στην προηγούμενη περίπτωση(συνάρτηση Rosenbrock) χρησιμοποιήθηκε κοινός συντελεστής BW όλους τους συνδυασμούς παραμέτρων. Οι τιμές που έλαβε είναι από 0.5 έως 0.1.

NI	HMS	HMCR	PAR	ΜΕΣΟΣ ΟΡΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ(10 ΕΚΤΕΛΕΣΕΙΣ)
500	10	0.6	0.1	0.260362
			0.5	0.694177
		0.9	0.1	0.670965
			0.5	0.91615
	30	0.6	0.1	0.688923
			0.5	0.647777
		0.9	0.1	0.568763
			0.5	0.512404
	50	0.6	0.1	0.591383
			0.5	0.57548
		0.9	0.1	0.61849
			0.5	0.163261
2000	10	0.6	0.1	0.021668
			0.5	0.000853
		0.9	0.1	0.333363
			0.5	0.181719
	30	0.6	0.1	0.009144
			0.5	0.005689
		0.9	0.1	0.091981
			0.5	0.001405
	50	0.6	0.1	0.016715

			0.5	0.007811
		0.9	0.1	0.003726
			0.5	0.091483
5000	10	0.6	0.1	0.00053
			0.5	1.6E-05
		0.9	0.1	0.000147
			0.5	3E-06
	30	0.6	0.1	0.00085
			0.5	3.5E-05
		0.9	0.1	4.7E-05
			0.5	2E-06
	50	0.6	0.1	0.00076
			0.5	6.5E-05
		0.9	0.1	0.00011
			0.5	1.2E-05

Από τον παραπάνω πίνακα παρατηρείται ότι σε κάθε πλήθος επαναλήψεων το βέλτιστο αποτέλεσμα υπολογίζεται όταν ο συντελεστής PAR λαμβάνει τιμή 0,5. Κατά τα άλλα, στις 500 επαναλήψεις η μνήμη χωρητικότητας 50 αρμονιών παρουσιάζει το καλύτερο αποτέλεσμα ενδεχομένως και λόγω της έντονης επίδρασης της αρχικοποίησης της μνήμης με τυχαίες τιμές. Στις 5000 επαναλήψεις τα αποτελέσματα είναι πολύ καλά σχεδόν για κάθε συνδυασμό παραμέτρων καθώς το χειρότερο αποτέλεσμα πετυχαίνει το ολικό ελάχιστο με ακρίβεια 3 δεκαδικών ψηφίων.

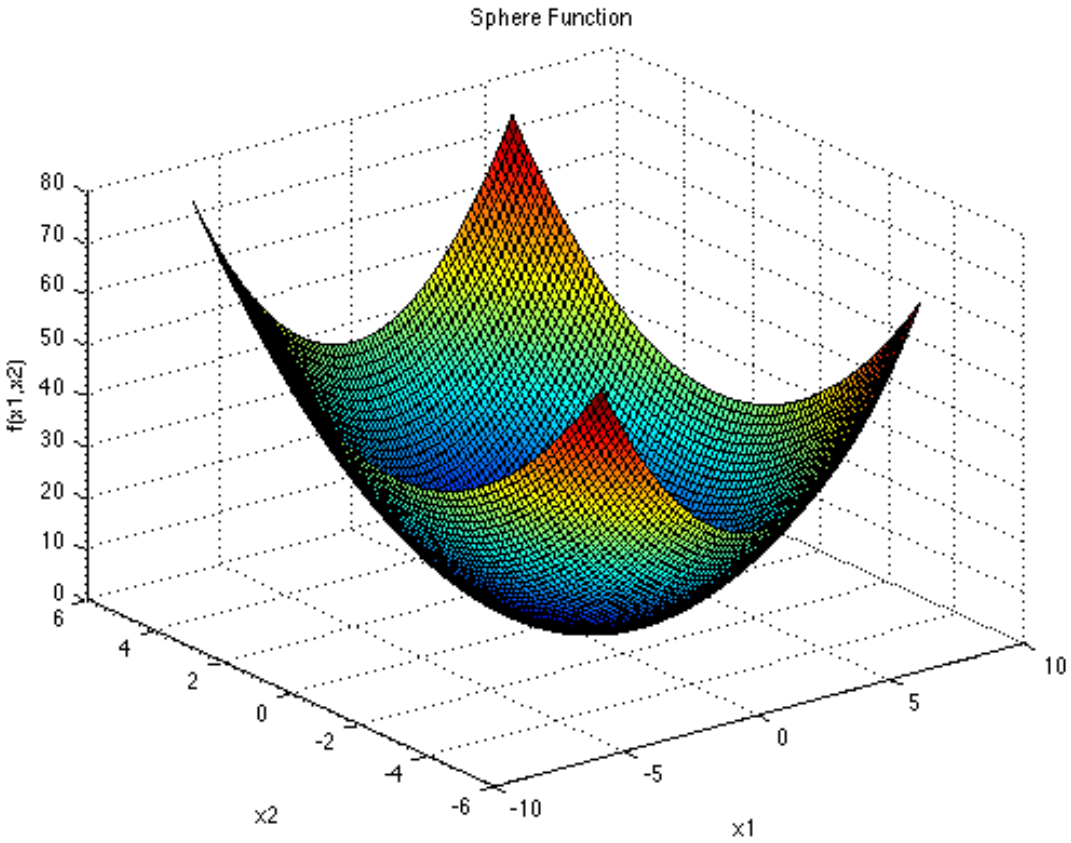
Η συνάρτηση Sphere

Η συνάρτηση Sphere είναι γνωστή και ως συνάρτηση De Jong, καθώς ήταν η πρώτη συνάρτηση που προτάθηκε από τον Kenneth A. de Jong. Η συνάρτηση είναι συμμετρική, κυρτή και τα τοπικά ελάχιστα είναι ίσα με τον αριθμό των διαστάσεων. Η βελτιστοποίηση της συνάρτησης θεωρείται γενικά εύκολη.

Η συνάρτηση Sphere ορίζεται ως εξής:

$$f(\mathbf{x}) = \sum_{i=1}^{d} x_i^2$$

Το ολικό ελάχιστο της συνάρτησης είναι **f(x) = 0** για $\mathbf{x_i = 0}$ με πεδίο ορισμού $\mathbf{-5.12 \leq x_i \leq 5.12}$



https://www.sfu.ca/~ssurjano/spheref.png

Ο αλγόριθμος δοκιμάστηκε στη συνάρτηση Sphere δύο διαστάσεων. Στις δοκιμές, χρησιμοποιήθηκαν διαφορετικοί συνδυασμοί παραμέτρων για να μελετηθεί καλύτερα η

συμπεριφορά του αλγορίθμου. Η παράμετρος BW είναι ίδια για κάθε εκτέλεση του προγράμματος με τιμές που κυμαίνονται από 0.5 έως 0.0005

NI	HMS	HMCR	PAR	ΜΕΣΟΣ ΟΡΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ(10 ΕΚΤΕΛΕΣΕΙΣ)
500	10	0.6	0.1	0.001456
			0.5	5.9E-05
		0.9	0.1	4,6E-05
			0.5	7E-06
	30	0.6	0.1	0.00509
			0.5	0.000635
		0.9	0.1	0.001084
			0.5	2.2E-05
	50	0.6	0.1	0.007738
			0.5	0.009944
		0.9	0.1	0.003461
			0.5	0.000275
2000	10	0.6	0.1	5E-06
			0.5	1E-06
		0.9	0.1	1E-06
			0.5	0
	30	0.6	0.1	1.7E-05
			0.5	3E-06
		0.9	0.1	2E-06
			0.5	0
	50	0.6	0.1	3.1E-05

			0.5	9E-06
		0.9	0.1	4E-06
			0.5	0
5000	10	0.6	0.1	1E-06
			0.5	0
		0.9	0.1	0
			0.5	0
	30	0.6	0.1	2E-06
			0.5	0
		0.9	0.1	0
			0.5	0
	50	0.6	0.1	1E-06
			0.5	0
		0.9	0.1	0
			0.5	0

Όπως ήταν αναμενόμενο, ο αλγόριθμος είναι ικανός να βρει το ολικό ελάχιστο ακόμα και στις 500 επαναλήψεις. Ενώ για περισσότερες επαναλήψεις το ελάχιστο εντοπίζεται με ακρίβεια άνω των 6 ψηφίων και πολλούς διαφορετικούς συνδυασμού παραμέτρων. Εύκολα παρατηρείται πως όταν οι συντελεστές HMCR και PAR λαμβάνουν μεγάλες τιμές ο αλγόριθμος έχει πάντα καλή απόδοση.

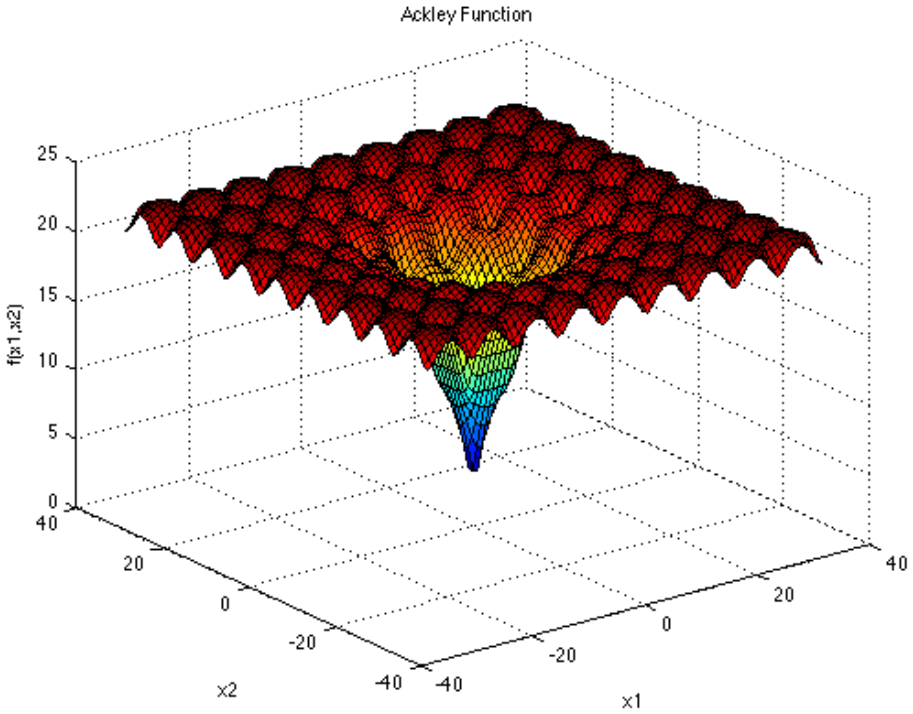
Η συνάρτηση Ackley

Η συνάρτηση Ackley χρησιμοποιείται ευρέως για την δοκιμή αλγορίθμων βελτιστοποίησης. Χαρακτηρίζεται από μια σχεδόν επίπεδη περιοχή με πολλαπλά τοπικά βέλτιστα και στο κέντρο της περιοχής αυτής σχηματίζεται μια μεγάλη "τρύπα" που οδηγεί στο ολικό βέλτιστο. Η συνάρτηση θεωρείται δύσκολη καθώς οι αλγόριθμοι συχνά παγιδεύονται σε κάποια από τα τοπικά βέλτιστα που σχηματίζονται.

Η συνάρτηση Ackley ορίζεται ως εξής:

$$f(\mathbf{x}) = -a\exp\left(-b\sqrt{\frac{1}{d}\sum_{i=1}^{d}x_i^2}\right) - \exp\left(\frac{1}{d}\sum_{i=1}^{d}\cos(cx_i)\right) + a + \exp(1)$$

Όπου **a = 20**, **b = 0.2** και **c = 2π.**
Το ολικό ελάχιστο της συνάρτησης είναι **f(x) = 0** για $\mathbf{x_i = 0}$ με πεδίο ορισμού $\mathbf{-32.768 \leq x_i \leq 32.768}$



https://www.sfu.ca/~ssurjano/ackley.png

Ο αλγόριθμος δοκιμάστηκε στη συνάρτηση Ackley δύο διαστάσεων. Στις δοκιμές, χρησιμοποιήθηκαν διαφορετικοί συνδυασμοί παραμέτρων για να μελετηθεί καλύτερα η

συμπεριφορά του αλγορίθμου. Η παράμετρος BW είναι ίδια για κάθε εκτέλεση του προγράμματος με τιμές που κυμαίνονται από 0.05 έως 1.5.

NI	HMS	HMCR	PAR	ΜΕΣΟΣ ΟΡΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ(10 ΕΚΤΕΛΕΣΕΙΣ)
500	10	0.6	0.1	1.065047
			0.5	0.086594
		0.9	0.1	0.780874
			0.5	0.047352
	30	0.6	0.1	1.591258
			0.5	0.995323
		0.9	0.1	0.941703
			0.5	0.57854
	50	0.6	0.1	2.028925
			0.5	2.947498
		0.9	0.1	1.572496
			0.5	0.433311
2000	10	0.6	0.1	0.112165
			0.5	0.044309
		0.9	0.1	0.023432
			0.5	0.010798
	30	0.6	0.1	0.064855
			0.5	0.056244
		0.9	0.1	0.018475
			0.5	0.008537

	50	0.6	0.1	0.058529
			0.5	0.014423
		0.9	0.1	0.009028
			0.5	0.006533
5000	10	0.6	0.1	0.002914
			0.5	0.001146
		0.9	0.1	0.001152
			0.5	0.000418
	30	0.6	0.1	0.002849
			0.5	0.001033
		0.9	0.1	0.001258
			0.5	0.000366
	50	0.6	0.1	0.00407
			0.5	0.002416
		0.9	0.1	0.000921
			0.5	0.000568

Για την συνάρτηση Ackley ο αλγόριθμος κατάφερε να εντοπίσει προσεγγιστικά το ολικό ελάχιστο με ακρίβεια 3 δεκαδικών ψηφίων. Για κάθε πλήθος επαναλήψεων οι παράμετροι αποδίδουν καλά για HMCR=0.9 και PAR-0.5. Αντίθετα, το μέγεθος της Αρμονικής Μνήμης που προσφέρει τα καλύτερα αποτελέσματα διαφέρει σε κάθε πλήθος επαναλήψεων.

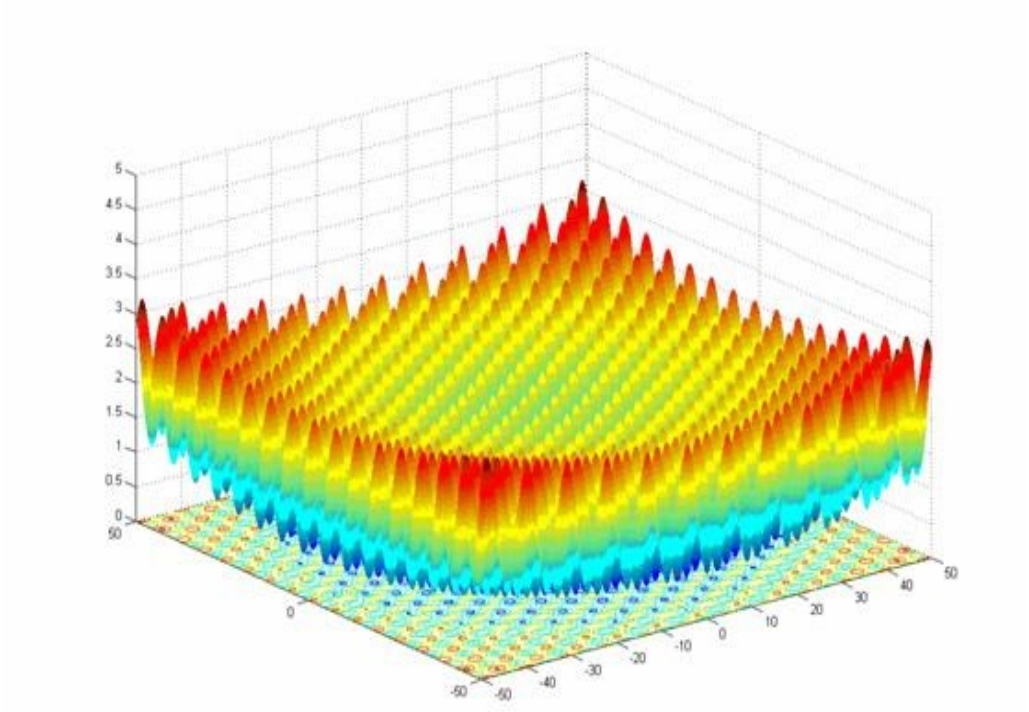
Η συνάρτηση Griewank

Η συνάρτηση Griewank είναι μια ακόμα συνάρτηση που χρησιμοποιείται ευρέως για την δοκιμή μεθόδων βελτιστοποίησης. Διαθέτει πολλά τοπικά βέλτιστα τα οποία κατανέμονται ομαλά και συμμετρικά σε όλο το πεδίο ορισμού.

Η συνάρτηση Griewank ορίζεται ως εξής:

$$f(\mathbf{x}) = \sum_{i=1}^{d} \frac{x_i^2}{4000} - \prod_{i=1}^{d} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$$

Το ολικό ελάχιστο της συνάρτησης είναι **f(x) = 0** για $\mathbf{x_i = 0}$ με πεδίο ορισμού $\mathbf{-600 \leq x_i \leq 600}$



http://www-optima.amp.i.kyoto-u.ac.jp/member/student/hedar/Hedar_files/TestGO_files/image8891.jpg

Ο αλγόριθμος δοκιμάστηκε στη συνάρτηση Griewank δύο διαστάσεων. Στις δοκιμές, χρησιμοποιήθηκαν διαφορετικοί συνδυασμοί παραμέτρων για να μελετηθεί καλύτερα η συμπεριφορά του αλγορίθμου. Η παράμετρος BW είναι ίδια για κάθε εκτέλεση του προγράμματος με τιμές που κυμαίνονται από 0.5 έως 15.

NI	HMS	HMCR	PAR	ΜΕΣΟΣ ΟΡΟΣ ΑΠΟΤΕΛΕΣΜΑΤΩΝ(10 ΕΚΤΕΛΕΣΕΙΣ)
500	10	0.6	0.1	0.147955
			0.5	0.146868
		0.9	0.1	0.18948
			0.5	0.332881
	30	0.6	0.1	0.258124
			0.5	0.206585
		0.9	0.1	0.17571
			0.5	0.221434
	50	0.6	0.1	0.415213
			0.5	0.284594
		0.9	0.1	0.241517
			0.5	0.197092
2000	10	0.6	0.1	0.031326
			0.5	0.025435
		0.9	0.1	0.056493
			0.5	0.035727
	30	0.6	0.1	0.044562
			0.5	0.02415

		0.9	0.1	0.061362
			0.5	0.033335
	50	0.6	0.1	0.03189
			0.5	0.036772
		0.9	0.1	0.059163
			0.5	0.031218
5000	10	0.6	0.1	0.028838
			0.5	0.019241
		0.9	0.1	0.040287
			0.5	0.001416
	30	0.6	0.1	0.021248
			0.5	0.02349
		0.9	0.1	0.032498
			0.5	0.00593
	50	0.6	0.1	0.023228
			0.5	0.012611
		0.9	0.1	0.056552
			0.5	0.004612

Η συνάρτηση Griewank ήταν η δυσκολότερη στην οποία δοκιμάστηκε ο αλγόριθμος. Μια πιθανή αιτία είναι ότι ορίστηκε πολύ μεγαλύτερο πεδίο ορισμού από τις προηγούμενες συναρτήσεις. Ωστόσο, και πάλι ο αλγόριθμος κατάφερε να εντοπίσει τιμές που προσεγγίζουν το ολικό βέλτιστο με ακρίβεια 3 δεκαδικών ψηφίων. Σε κάθε περίπτωση, παρατηρείται ότι ο αλγόριθμος αποδίδει καλά όταν ο συντελεστής PAR λαμβάνει μεγάλες τιμές. Αντίθετα για τον συντελεστή HMCR παρατηρείται ότι σε μικρό πλήθος επαναλήψεων είναι καλύτερο να λαμβάνει μικρές τιμές για να μπορεί να εξερευνηθεί το μεγάλο πεδίο ορισμού.

Συγκεντρωτικός πίνακας αποτελεσμάτων

	Rosenbrock	Rastrigin	Sphere	Ackley	Griewank
NI=500	HMS=30	HMS=50	HMS=10	HMS=10	HMS=10
	HMCR=0.6	HMCR=0.9	HMCR=0.9	HMCR=0.9	HMCR=0.6
	PAR=0.1	PAR=0.5	PAR=0.5	PAR=0.5	PAR=0.5
	0.060309	**0.163261**	**7E-06**	**0.047352**	**0.146868**
NI=2000	HMS=10	HMS=10	HMS=*	HMS=50	HMS=50
	HMCR=0.9	HMCR=0.6	HMCR=0.9	HMCR=0.9	HMCR=0.6
	PAR=0.5	PAR=0.5	PAR=0.5	PAR=0.5	PAR=0.5
	0.006847	**0.000853**	**0**	**0.006533**	**0.036772**
NI=5000	HMS=10	HMS=30	HMS=*	HMS=30	HMS=10
	HMCR=0.9	HMCR=0.9	HMCR=*	HMCR=0.9	HMCR=0.9
	PAR=0.5	PAR=0.5	PAR=*	PAR=0.5	PAR=0.5
	0.000173	**2E-06**	**0**	**0.000366**	**0.001416**

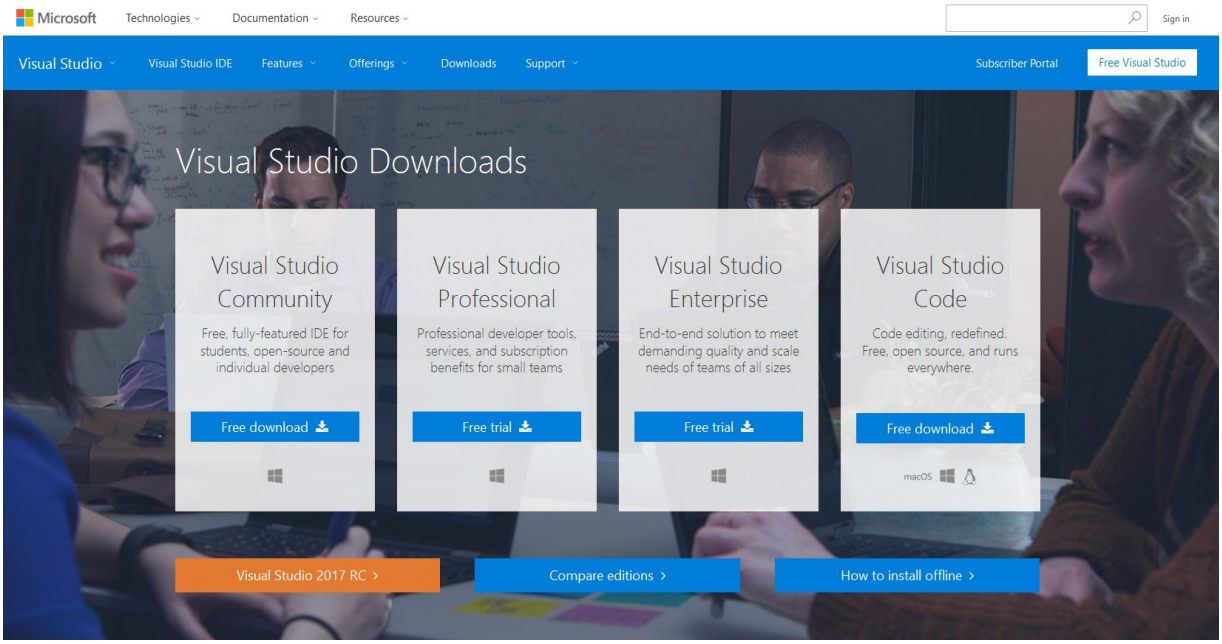
Στον παραπάνω πίνακα αποτυπώνονται συνοπτικά τα καλύτερα αποτελέσματα του αλγορίθμου για κάθε μαθηματική συνάρτηση για κάθε πλήθος επαναλήψεων. Στις 500 επαναλήψεις, οι παράμετροι που έδωσαν τα κατά μέσο όρο τα καλύτερα αποτελέσματα είναι HMS=10, HMCR=0.9 και PAR=0.5. Στις 2000 επαναλήψεις, οι καλύτεροι συνδυασμοί παραμέτρων είναι HMS=10 ή HMS=50, HMCR=0.9 και PAR=0.5. Τέλος, για τις 5000 επαναλήψεις οι καλύτεροι παράμετροι είναι HMS=10 ή HMS=30, HMCR=0.9 και PAR=0.5.

VIII. ΤΟ ΠΡΟΓΡΑΜΜΑ

Επισκόπηση

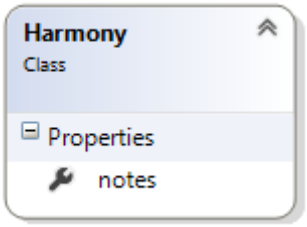
Η υλοποίηση του προγράμματος βασίστηκε στην γλώσσα προγραμματισμού **C#**. Η C# είναι μια αντικειμενοστραφής γλώσσα υψηλού επιπέδου που αναπτύσσεται από την Microsoft, και βασίζεται στο .NET framework το οποίο επίσης ανήκει στην Microsoft. Ένα πολύ σημαντικό χαρακτηριστικό της C#, είναι ότι ο κώδικας είναι διαχειριζόμενος(managed). Αυτό σημαίνει ότι κατά την εκτέλεση του προγράμματος, ο κώδικας μεταφράζεται σε ένα ενδιάμεσο επίπεδο και κατόπιν σε γλώσσα μηχανής. Αυτή η προσέγγιση αποφέρει και πλεονεκτήματα(αυτόματη διαχείριση μνήμης, συμβατότητα) και μειονεκτήματα(ταχύτητα εκτέλεσης, μέγεθος).

Για την υλοποίηση του προγράμματος χρησιμοποιήθηκε επίσης το Ολοκληρωμένο Περιβάλλον Ανάπτυξης(Integrated Development Environment – IDE) **Visual Studio 2015**, το οποίο αναπτύσσεται από την Microsoft. Η Community έκδοση του Visual Studio διατίθεται δωρεάν στον σύνδεσμο **https://www.visualstudio.com/downloads/**, και τρέχει μόνο σε λειτουργικά συστήματα Windows. Οι υπόλοιπες εκδόσεις είναι επί πληρωμή.



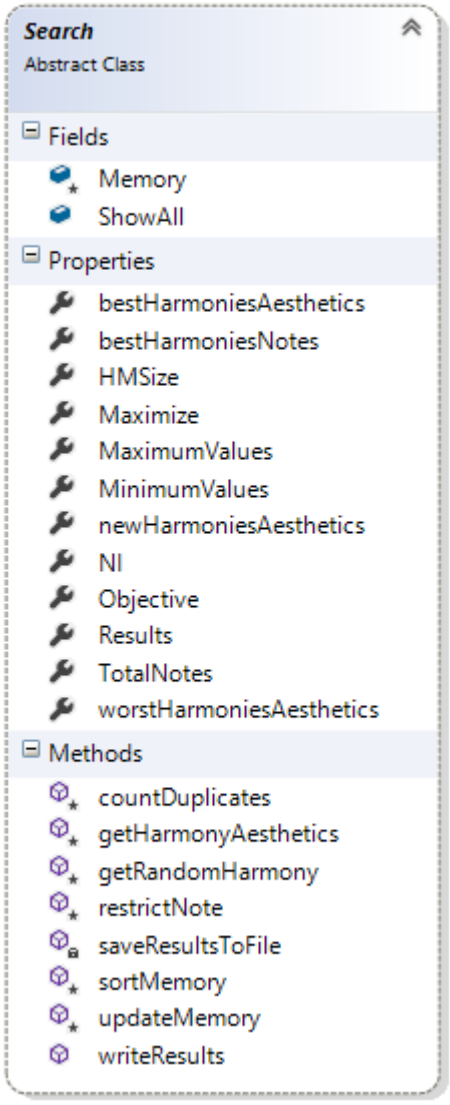
Τεκμηρίωση Κώδικα

➢ Η κλάση Harmony



Η κλάση Harmony περιγράφει την ανατομία και την δομή μιας Αρμονίας, δηλαδή μιας υποψήφιας λύσης. Αποτελείται μονάχα από μια ιδιότητα που είναι ένα διάνυσμα τύπου double και περιέχει τις μεταβλητές απόφασης(Νότες). Τα αντικείμενα αυτής της κλάσης χρησιμοποιούνται σε κάθε αλγόριθμο Harmony Search.

➢ Η κλάση Search



Η κλάση Search είναι η μητρική κλάση από την οποία κληρονομούν όλες οι κλάσεις στις οποίες υλοποιείται κάποια παραλλαγή του αλγορίθμου Harmony Search. Σε αυτή την κλάση υλοποιούνται κάποιες συναρτήσεις που είναι κοινές σε όλες τις παραλλαγές της Harmony Search. Η κλάση Search δεν χρησιμοποιείται πουθενά άμεσα ή ανεξάρτητα, ωστόσο είναι πολύ χρήσιμη διότι αποφεύγεται η επαναλαμβανόμενη συγγραφή παρόμοιων, ή και πανομοιότυπων μεθόδων. Αυτό σημαίνει, ότι είναι αρκετά ευκολότερο να εντοπιστούν πιθανά λογικά σφάλματα(bugs) του προγράμματος. Ταυτόχρονα, καθιστά ευκολότερη την προσθήκη νέων χαρακτηριστικών και δυνατοτήτων σε οποιαδήποτε παραλλαγή της Harmony Search.

getHarmonyAesthetics: Σε αυτήν την μέθοδο υπολογίζεται η ποιότητα της κάθε υποψήφιας λύσης. Η μέθοδος δέχεται σαν είσοδο ένα αντικείμενο τύπου Harmony και επιστρέφει μια τιμή τύπου double.

getRandomHarmony: Σε αυτήν την μέθοδο επιστρέφεται μια τυχαία Αρμονία(μια υποψήφια λύση δηλαδή) από αυτές που είναι αποθηκευμένες στην Αρμονική Μνήμη. Χρησιμοποιείται από τον μηχανισμό HMCR.

restrictNote: Η μέθοδος αυτή ελέγχει αν μια νότα(μεταβλητή απόφασης) βρίσκεται εκτός πεδίου ορισμού. Αν ξεπερνάει το μέγιστο όριο τότε αντικαθίσταται με την μέγιστη τιμή ώστε να είναι εντός των επιτρεπτών συνόρων. Αντίστοιχα, το ίδιο συμβαίνει όταν η τιμή της νότας είναι

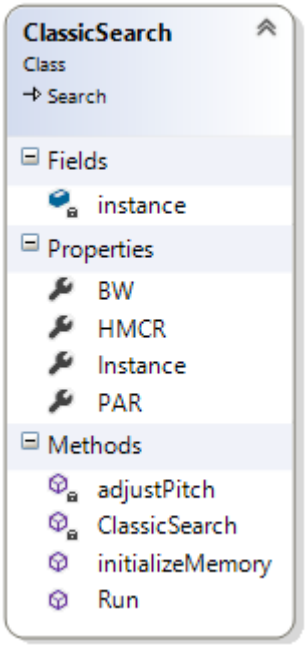
μικρότερη από το ελάχιστο όριο. Αν είναι εντός ορίων τότε η μέθοδος επιστρέφει χωρίς να πράξει το παραμικρό.

sortMemory: Σε αυτή την συνάρτηση ταξινομείται το collection της Αρμονικής Μνήμης(HM) με βάση την ποιότητα της εκάστοτε υποψήφιας λύσης. Η ταξινόμηση γίνεται είτε κατά αύξουσα είτε κατά φθίνουσα σειρά, ανάλογα με την επιλογή που έχει κάνει ο χρήστης.

updateMemory: Αυτή η μέθοδος ενημερώνει την Αρμονική Μνήμη. Αν η νέα υποψήφια λύση(Αρμονία) είναι καλύτερη από την χειρότερη υποψήφια λύση της Αρμονικής Μνήμης τότε την αντικαθιστά και καλεί την μέθοδο sortMemory(). Διαφορετικά επιστρέφει.

writeResults: Σε αυτή την μέθοδο τυπώνονται όλες οι Αρμονικές Μνήμες όλων των επαναλήψεων. Είναι μια χρονοβόρα διαδικασία που όμως είναι χρήσιμη για να μελετηθεί καλύτερα ο αλγόριθμος. Ο χρήστης μπορεί να επιλέξει αν θα εκτελεστεί.

➢ Η κλάση ClassicSearch



Η κλάση ClassicSearch υλοποιεί την κλασσική έκδοση του αλγορίθμου Harmony Search. Κληρονομεί τα χαρακτηριστικά και τις μεθόδους της κλάσης Search. Όταν ο χρήστης επιλέγει τον κλασσικό Harmony Search για να βελτιστοποιήσει μια συνάρτηση τότε δημιουργείται ένα αντικείμενο αυτής της κλάσης. Οι ιδιότητες **BW**, **HMCR** και **PAR** λαμβάνουν τις τιμές που θα τους δώσει ο χρήστης. Όπως αναλύθηκε και σε προηγούμενα κεφάλαια αυτοί οι συντελεστές είναι απαραίτητοι για να λειτουργήσει σωστά ο αλγόριθμος.

Instance: Πρόκειται για ένα property μέσω του οποίου υλοποιείται το Singleton Pattern. Αυτό σημαίνει, πως για αυτήν την κλάση δεν μπορούν να δημιουργηθούν παραπάνω από ένα αντικείμενα.

adjustPitch: Η μέθοδος adjustPitch υλοποιεί τον μηχανισμό **PAR** του αλγορίθμου. Διαφέρει σημαντικά ανάλογα με την παραλλαγή που χρησιμοποιείται.

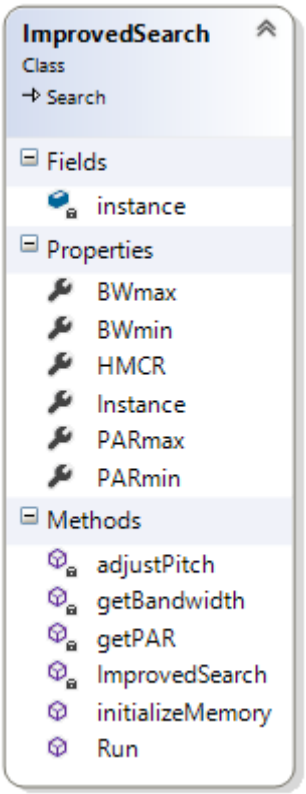
initializeMemory: Αυτή η μέθοδος αρχικοποιεί τον αλγόριθμο. Στην ουσία μηδενίζει την Αρμονική Μνήμη και την ξαναγεμίζει με τυχαίες αρμονίες οι οποίες με τη σειρά τους λαμβάνουν τυχαίες τιμές. Είναι χρήσιμη όταν πρέπει να τρέξει η αναζήτηση πολλές φορές προκειμένου να εξαχθούν αξιόπιστα συμπεράσματα.

Run: Σε αυτή τη μέθοδο υλοποιείται όλη η στρατηγική και η λογική του αλγορίθμου. Είναι μια αφηρημένη μέθοδος που καθοδηγεί την αναζήτηση απομονώνοντας την υλοποίηση

χαμηλότερου επιπέδου διαδικασιών οι οποίες όμως καλούνται στο σώμα της αυτής της μεθόδου.

**Οι παραπάνω συναρτήσεις υλοποιούνται με παρόμοια λογική σε κάθε παραλλαγή της Harmony Search. Κατά συνέπεια, στην ανάλυση των επόμενων κλάσεων δεν θα τεκμηριωθούν εκ νέου(αν υπάρχουν). Θα αναλύονται μόνο όταν παρουσιάζουν σημαντικές διαφορές σε σχέση με την κλασσική Harmony Search.*

➢ Η κλάση ImprovedSearch

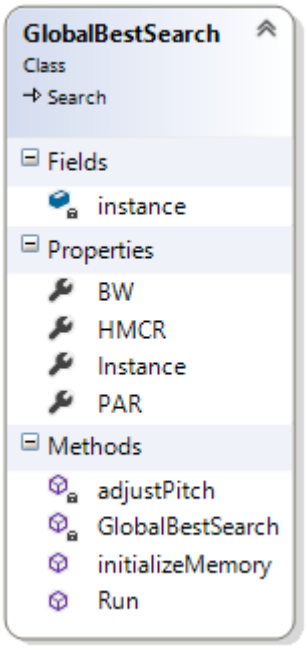


Η κλάση ImprovedSearch υλοποιεί την βελτιωμένη έκδοση του αλγορίθμου Harmony Search. Διαφέρει από τον κλασσικό Harmony Search στον μηχανισμό PAR ο οποίος αυξάνεται με το πέρασμα του χρόνου. Ο χρήστης ορίζει μονάχα την ελάχιστη(**PARmin**) και την μέγιστη(**PARmax**) πιθανότητα PAR. Μια ακόμη σημαντική διαφορά αφορά τον συντελεστή BW, ο οποίος στην συγκεκριμένη παραλλαγή μειώνεται εκθετικά με την πάροδο του χρόνου. Ομοίως, ο χρήστης αρκεί να ορίσει το μέγιστο(**BWmax**) και το ελάχιστο(**BWmin**) όριο του συντελεστή BW.

getPAR: Καλείται κάθε φορά που υλοποιείται ο μηχανισμός HMCR και επιστρέφει την πιθανότητα PAR ανάλογα με τα όρια που έχει θέσει ο χρήστης.

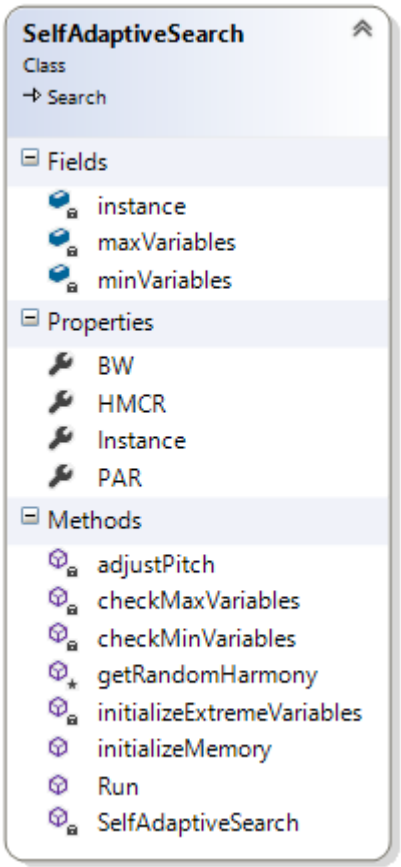
getBandwidth: Κάθε φορά που υλοποιείται ο μηχανισμός PAR καλείται αυτή η συνάρτηση. Η τιμή που θα επιστρέψει εξαρτάται από τα όρια που έχει θέσει ο χρήστης και από τον αριθμό επανάληψης στον οποίο βρίσκεται ο αλγόριθμος.

➢ Η κλάση GlobalBestSearch



Η κλάση GlobalBestSearch υλοποιεί την επονομαζόμενη "συνολικά βέλτιστη" έκδοση του αλγορίθμου Harmony Search. Διαφέρει από τις υπόλοιπες παραλλαγές διότι δεν χρησιμοποιείται καθόλου η παράμετρος **BW**. Η αυξομείωση της εκάστοτε μεταβλητής απόφασης γίνεται με βάση τις τιμές της κορυφαίας, ως εκείνη τη στιγμή, λύσης που είναι αποθηκευμένη στην Αρμονική Μνήμη. Πρακτικά, αυτό σημαίνει ότι η μέθοδος **adjustPitch** έχει διαφορετική λογική συγκριτικά με τις υπόλοιπες παραλλαγές. Κατά τα άλλα, ο μηχανισμός **HMCR** έχει ακριβώς την ίδια φιλοσοφία που είχε και στις προηγούμενες παραλλαγές. Κάποια από τα χαρακτηριστικά της GlobalBestSearch και κάποιες από τις μεθόδους της κληρονομούνται από την γονική κλάση **Search**.

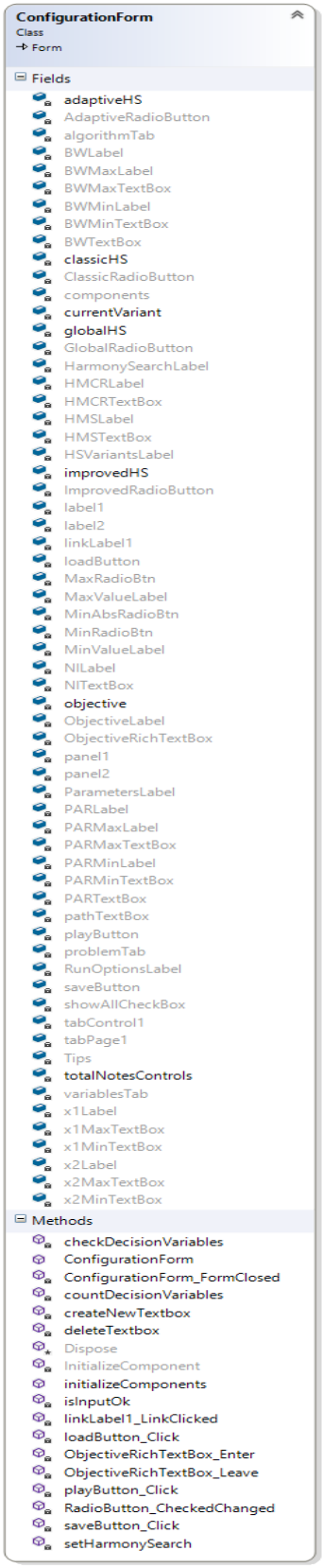
➢ Η κλάση SelfAdaptiveSearch



Η κλάση **SelfAdaptiveSearch** υλοποιείται η λογική της Αυτοπροσαρμοστικής Harmony Search. Ξεχωρίζει από τις υπόλοιπες παραλλαγές ως προς την μέθοδο adjustPitch(). Σε αντίθεση με τις προηγούμενες παραλλαγές, σε αυτή την περίπτωση ο μηχανισμός **PAR** εκτελείται λαμβάνοντας υπόψιν την μέγιστη ή την ελάχιστη μεταβλητή απόφασης της Αρμονικής Μνήμης. Αυτό σημαίνει ότι θα πρέπει κάθε αντικείμενο αυτής της κλάσης να διαθέτει έναν τύπο δεδομένων για την αποθήκευση των μέγιστων μεταβλητών απόφασης(**maxVariables**) και αντίστοιχα, άλλον ένα ίδιο τύπο δεδομένων για τις ελάχιστες τιμές(**minVariables**). Όπως και στην κλάση GlobalBestSearch έτσι και εδώ ο συντελεστής BW δεν χρησιμοποιείται καθόλου.

initializeExtremeVariables: Η μέθοδος αυτή, καλείται στην αρχή της Αναζήτησης και αρχικοποιεί τις μεταβλητές maxVariables και minVariables την πρώτη φορά όταν η Αρμονική Μνήμη είναι γεμάτη με τυχαίες αρμονίες.

checkMaxVariables: Καλείται στο τέλος κάθε γενιάς και ελέγχει αν ενημερώθηκε η Αρμονική Μνήμη με κάποια νέα Αρμονία. Αν ενημερώθηκε, τότε ελέγχεται αν υπάρχει νέα μέγιστη μεταβλητή απόφασης. Αν υπάρχει τότε ενημερώνεται η μεταβλητή **maxVariables**.



checkMaxVariables: Η λογική της είναι ίδια με αυτή της checkMaxVariables() με την διαφορά ότι εδώ ελέγχεται αν υπάρχει νέα ελάχιστη μεταβλητή απόφασης. Αν υπάρχει τότε ενημερώνεται η μεταβλητή **minVariables**.

➢ Η κλάση ConfigurationForm

Η κλάση ConfigurationForm είναι η κλάση που είναι υπεύθυνη για το UI της φόρμας στην οποία ο χρήστης ρυθμίζει της παραμέτρους του αλγορίθμου. Κληρονομεί από την κλάση **Form** η οποία είναι μια κλάση που παρέχεται από το .NET framework. Τα πεδία αυτής της κλάσης αποτελούνται κυρίως από παραμέτρους που έχει η κάθε παραλλαγή του αλγορίθμου Harmony Search ή και άλλα πεδία που είναι κοινά για όλες τις υλοποιήσεις της Harmony Search. Συνήθως κάθε παράμετρος αποτελείται από ένα **Label** και ένα **TextBox**. Στο Label φαίνεται η ονομασία της εκάστοτε παραμέτρου, και στο TextBox το περιεχόμενο της κάθε παραμέτρου, το οποίο ορίζεται από τον χρήστη. Ωστόσο, κατά την εκκίνηση του προγράμματος, τα TextBox περιέχουν προεπιλεγμένες τιμές και ο χρήστης μπορεί είτε να τις κρατήσει είτε να τις τροποποιήσει. Παράλληλα, υπάρχουν δύο ομάδες από **RadioButton**. Και οι δύο ομάδες περικλείονται από ένα **Panel** που καθιστά τις δύο ομάδες ανεξάρτητες. Στην πρώτη ομάδα, ο χρήστης επιλέγει αν η συνάρτηση που εξετάζεται πρόκειται να μεγιστοποιηθεί ή να ελαχιστοποιηθεί. Στην δεύτερη ομάδα, ο χρήστης διαλέγει τον αλγόριθμο που θα αναζητήσει την βέλτιστη τιμή της συνάρτησης. Label, TextBox, Panel και RadioButton είναι όλα είναι κλάσεις του συστήματος και δεν απαιτείται να γραφούν από την αρχή.

isInputOK: Σε αυτή την μέθοδο ελέγχεται αν οι τιμές που πληκτρολόγησε ο χρήστης στο κάθε πεδίο είναι σωστές. Αν υπάρχει λάθος τότε εμφανίζεται ένα ανάλογο μήνυμα.

playButton_Click: Εφόσον όλα τα πεδία έχουν αποδεκτές τιμές, τότε μόλις ο χρήστης κάνει αριστερό κλικ στο **playButton** θα οδηγηθεί στην επόμενη οθόνη που απεικονίζει τα αποτελέσματα του αλγορίθμου.

RadioButton_CheckedChanged: Είναι μια μέθοδος event handler που καλείται όταν ο χρήστης πλοηγείται ανάμεσα στις διάφορες παραλλαγές της Harmony Search. Ανάλογα με την επιλογή που είναι ενεργοποιημένη τότε εμφανίζονται και οι ανάλογοι παράμετροι λίγο πιο δίπλα.

setHarmonySearch: Καλείται από την μέθοδο playButton_Click(). Αφού πρώτα έχει ελεγχθεί ότι όλα τα πεδία είναι σε κατάλληλη μορφή, τότε αρχικοποιείται ο αλγόριθμος σύμφωνα πάντα με τις επιλογές που έχει κάνει ο χρήστης.

countDecisionVariables: Η μέθοδος αυτή μετράει τον πλήθος των μεταβλητών απόφασης της αντικειμενικής συνάρτησης που εισήγαγε ο χρήστης.

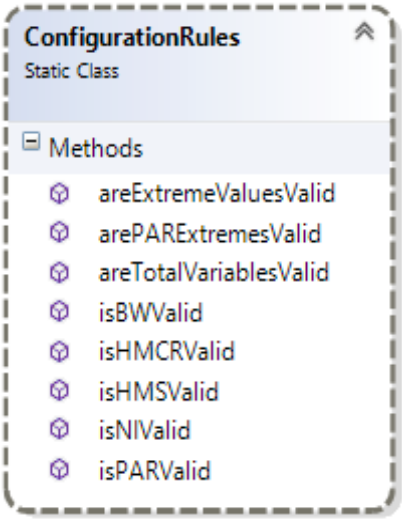
createNewTextbox: Αν η καταμέτρηση μεταβλητών απόφασης δείξει ότι υπάρχουν περισσότερες μεταβλητές απόφασης από τα αντίστοιχα textbox τότε δημιουργούνται δυναμικά νέα textbox. Στα textbox αυτά ορίζεται το πεδίο ορισμού της εκάστοτε μεταβλητής απόφασης.

deleteTextbox: Είναι η αντίστροφη της προηγούμενης μεθόδου. Όταν η καταμέτρηση μεταβλητών απόφασης δείξει ότι υπάρχουν περισσότερα textbox από αυτά που απαιτούνται τότε διαγράφονται τα περιττά textbox.

saveButton_Click: Αποθηκεύει όλες τις παραμέτρους που υπάρχουν στα πεδία σε ένα εξωτερικό αρχείο που θα βρίσκεται σε τοποθεσία που επέλεξε ο χρήστης..

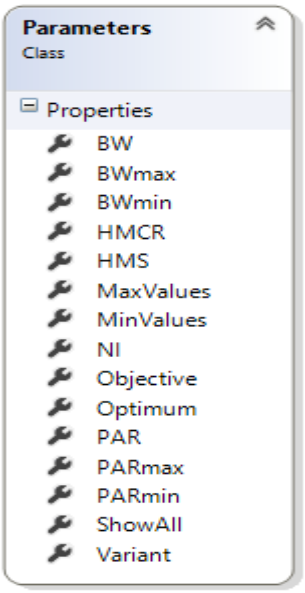
loadButton_Click: Φορτώνει όλες τις παραμέτρους από εξωτερικό αρχείο που βρίσκεται στη διεύθυνση που πληκτρολόγησε ο χρήστης.

➢ Η κλάση ConfigurationRules



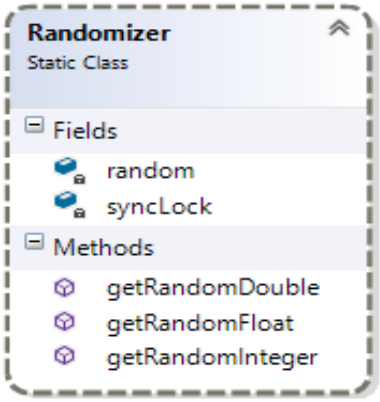
Η κλάση ConfigurationRules είναι μια **static** κλάση που χρησιμοποιείται για να ελεγχθούν οι τιμές που πληκτρολόγησε ο χρήστης στα πεδία των παραμέτρων του αλγορίθμου. Έχει μόνο μεθόδους και κανένα πεδίο ή χαρακτηριστικό. Συγκεκριμένα, ελέγχεται η εγκυρότητα των παραμέτρων **PAR, BW, HMCR, HMS, NI,** οι τιμές του πεδίου ορισμού και το πλήθος των μεταβλητών απόφασης. Όλες οι μέθοδοι επιστρέφουν τιμές τύπου bool καθώς χρησιμοποιούνται από αντικείμενα της κλάσης ConfigurationForm μέσα σε δομές επιλογής. Η κλάση είναι static, πράγμα που σημαίνει ότι δεν μπορούν να υπάρξουν αντικείμενα αυτής της κλάσης μέσα στο πρόγραμμα.

➢ Η κλάση Parameters



Η κλάση Parameters περιέχει όλες τις παραμέτρους που χρειάζεται ο αλγόριθμος για να λειτουργήσει. Χρησιμοποιείται μπορεί ο χρήστης να αποθηκεύει και να χρησιμοποιεί πολλές φορές τις ίδιες ρυθμίσεις. Κατά τις λειτουργίες save και load ένα αντικείμενο αυτής της κλάσης σειριοποιείται σε μορφή json για αποθήκευση ενώ στην φόρτωση γίνεται η αντίστροφη διαδικασία.

➢ Η κλάση Randomizer

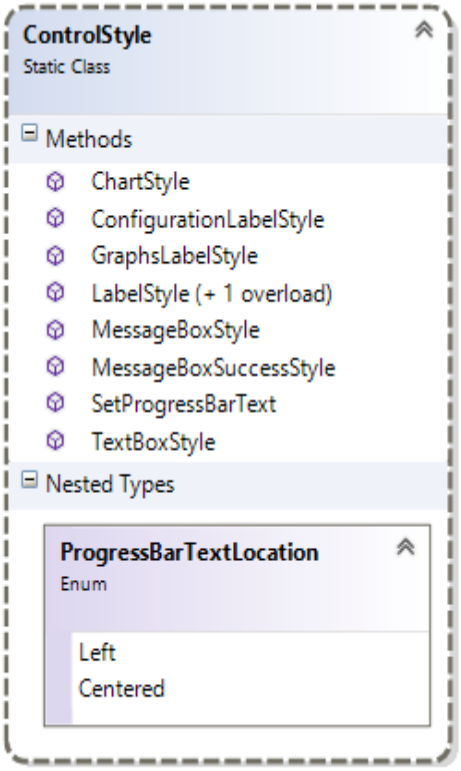


Η κλάση Randomizer είναι μια static κλάση που χρησιμοποιείται κατά τη διάρκεια της βελτιστοποίησης μιας συνάρτησης. Είναι πολύ χρήσιμη όχι μόνο για την Harmony Search αλλά και για σχεδόν κάθε Μεταευρετικό αλγόριθμο. Πολλοί μηχανισμοί της Harmony Search έχουν στοχαστικό χαρακτήρα και κατά συνέπεια η ύπαρξη κώδικα που θα παράγει τυχαίες τιμές είναι απαραίτητη.

➢ Η κλάση ControlStyle

Η κλάση ControlsStyle είναι static και περιέχει κώδικα και μεθόδους που έχουν σχέση με την εμφάνιση και την τοποθεσία διάφορων στοιχείων του UI μέσα στη φόρμα.

SetProgressBarText: Ορίζει δυναμικά το μήνυμα που θα αναγράφεται στην μπάρα αναμονής(progress bar) και υποδεικνύει πόσο σύντομα θα τελειώσει η βελτιστοποίηση.



TextBoxStyle: Ορίζει την θέση, το μέγεθος και την μορφή των textbox που δημιουργούνται δυναμικά. Συνήθως είναι textbox που έχουν σχέση με τις μεταβλητές απόφασης του προβλήματος.

ConfigurationLabelStyle: Ορίζεται το στυλ των label που αφορούν την ConfigurationForm.

GraphsLabelStyle: Ομοίως με την παραπάνω μέθοδο, ορίζονται τα στυλ των label που αφορούν την GraphsForm.

LabelStyle: Ορίζεται το στυλ των label όλου του κώδικα.

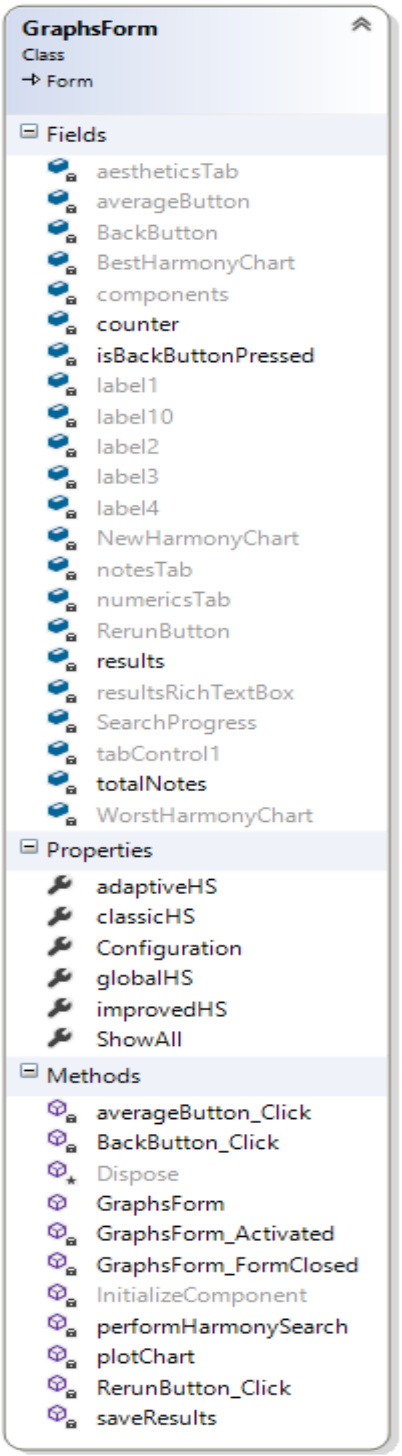
ChartStyle: Αφορά το στυλ των chart των μεταβλητών απόφασης της GraphsForm

➢ Η κλάση GraphsForm

Η κλάση GraphsForm είναι η κλάση που περιέχει την λογική του UI της δεύτερης φόρμας η οποία δείχνει τα αποτελέσματα της βελτιστοποίησης. Όπως και η κλάση ConfigurationForm κληρονομεί από την κλάση Form. Στο UI της φόρμας υπάρχει ένα TabControl με τρεις καρτέλες. Στην πρώτη καρτέλα εμφανίζονται τα αριθμητικά αποτελέσματα της βελτιστοποίησης και στις υπόλοιπες διάφορα σχεδιαγράμματα. Τα σχεδιαγράμματα έγιναν με χρήση του Chart, το οποίο είναι ένα control που παρέχεται από τις βιβλιοθήκες του .NET framework.

plotChart: Σε αυτή την μέθοδο υλοποιούνται τα σχεδιαγράμματα με τα αποτελέσματα της βελτιστοποίησης.

PerformHarmonySearch: Σε αυτήν την μέθοδο αρχικοποιείται και εν συνεχεία εκτελείται η παραλλαγή της Harmony Search που επιλέχθηκε. Καλείται από τα event handlers RerunButton_Click και GraphsForm_Activated.



BackButton_Click: Είναι event handler που καλείται όταν ο χρήστης πατάει το κουμπί Back. Η λειτουργία είναι να κλείνει την τρέχουσα φόρμα και να ανοίγει την ConfigurationForm.

RerunButton_Click: Αυτός ο event handler καλείται όταν γίνεται αριστερό κλικ στο κουμπί Rerun. Η λειτουργία είναι να αρχικοποιήσει τον αλγόριθμο με τις ίδιες παραμέτρους και να τον εκτελέσει εκ νέου.

GraphsForm_Activated: Είναι event handler που ενεργοποιείται όταν η GraphsForm εμφανίζεται στην οθόνη. Σε αυτή την μέθοδο γίνεται τρέχει ο αλγόριθμος.

GraphsForm_FormClosed: Είναι event handler και καλείται για να τερματίσει η εφαρμογή.

Η βιβλιοθήκη NCalc

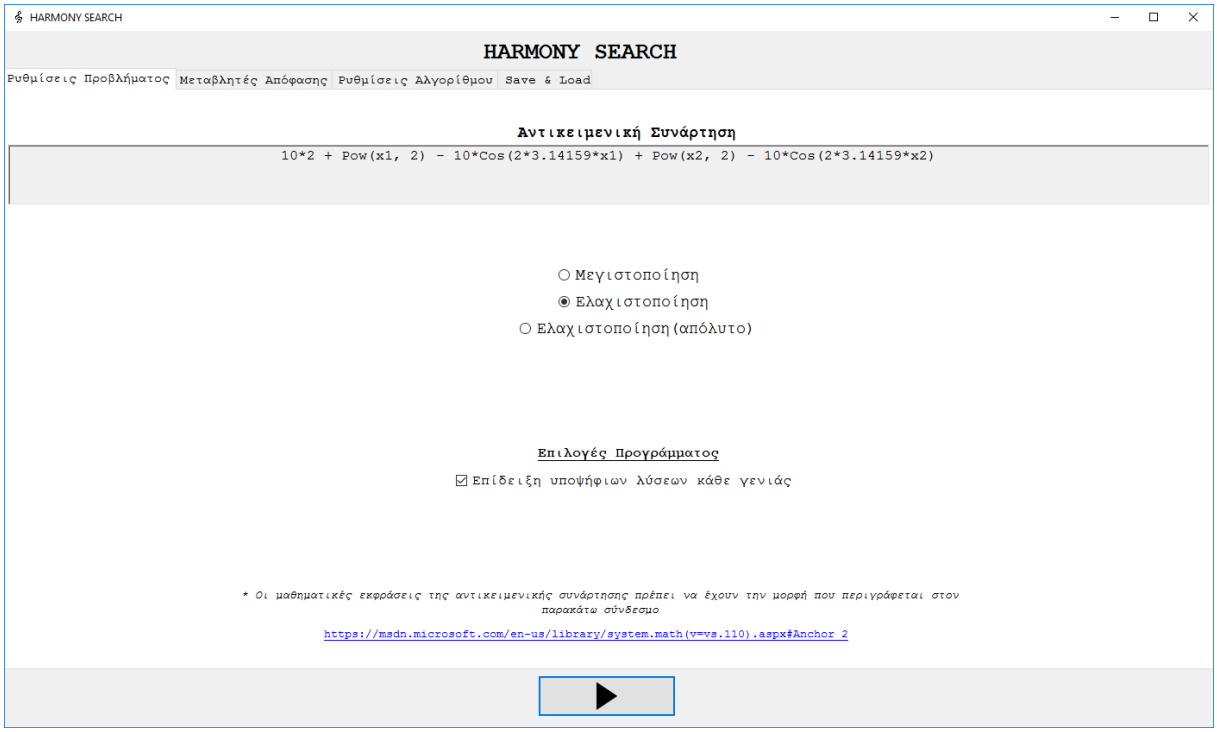
Η βιβλιοθήκη **NCalc** είναι ανοιχτού κώδικα(open source), και διατίθεται δωρεάν από την πλατφόρμα NuGet του Visual Studio, υπό το καθεστώς που επιβάλλει η MIT άδεια χρήσης. Επεξεργάζεται και υπολογίζει μαθηματικές εκφράσεις που είναι γραμμένες με την μορφή αλφαριθμητικών(strings). Η βιβλιοθήκη NCalc, δέχεται ένα string ως είσοδο, το αναλύει, και το μετατρέπει στην αντίστοιχη μαθηματική έκφραση. Η μετατροπή αυτή γίνεται με τη βοήθεια της κλάσης Math η οποία παρέχει constant και static μεθόδους για τριγωνομετρικές, λογαριθμικές και άλλες βασικές μαθηματικές συναρτήσεις. Η κλάση Math παρέχεται από το .NET framework. Η βιβλιοθήκη NCalc, έχει γραφτεί στη γλώσσα C#, ωστόσο μπορεί να χρησιμοποιηθεί και από άλλες γλώσσες προγραμματισμού χωρίς όμως να έχει την ίδια απόδοση. Η NCalc μπορεί να υπολογίσει οιαδήποτε μαθηματική έκφραση αρκεί να έχει συνταχθεί με τον κατάλληλο τρόπο. Στον σύνδεσμο https://msdn.microsoft.com/en-us/library/system.math(v=vs.110).aspx φαίνονται όλες οι συναρτήσεις της κλάσης Math με την αντίστοιχη τεκμηρίωση κάθε φορά.

Η βιβλιοθήκη NCalc χρησιμοποιήθηκε στο πρόγραμμα, για να μπορεί ο χρήστης να εισάγει κάθε φορά την αντικειμενική συνάρτηση της αρεσκείας του. Με αυτόν τον τρόπο, δίνεται η δυνατότητα να βελτιστοποιείται εύκολα οιαδήποτε μαθηματική συνάρτηση χωρίς να τροποποιείται ο πηγαίος κώδικας.

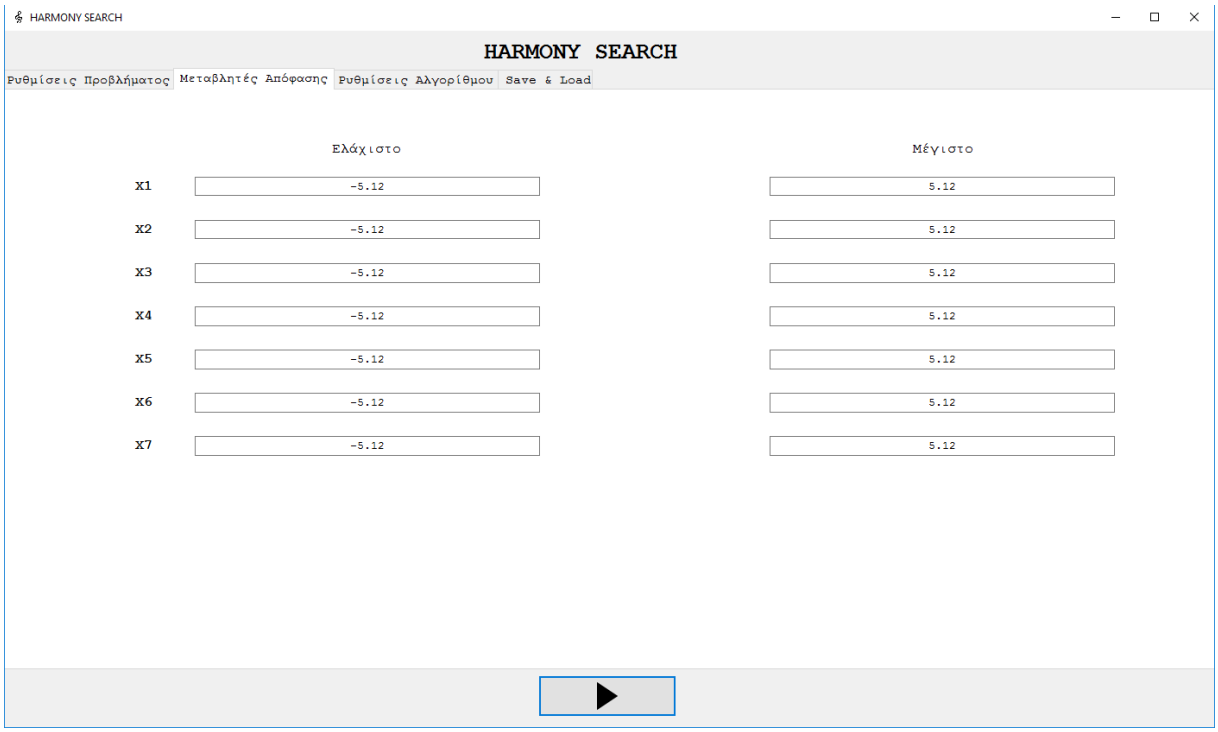
Σύστημα Ελέγχου Εκδόσεων

Ο πηγαίος κώδικας του προγράμματος είναι ανεβασμένος στην διαδικτυακή πλατφόρμα **GitHub.** Οποιοσδήποτε μπορεί να κατεβάσει τον κώδικα που βρίσκεται στον σύνδεσμο https://github.com/antogian/HarmonySearch, και να τον τροποποιήσει, καθώς έχει οριστεί η άδεια χρήσης **GPL(General Public License)**.

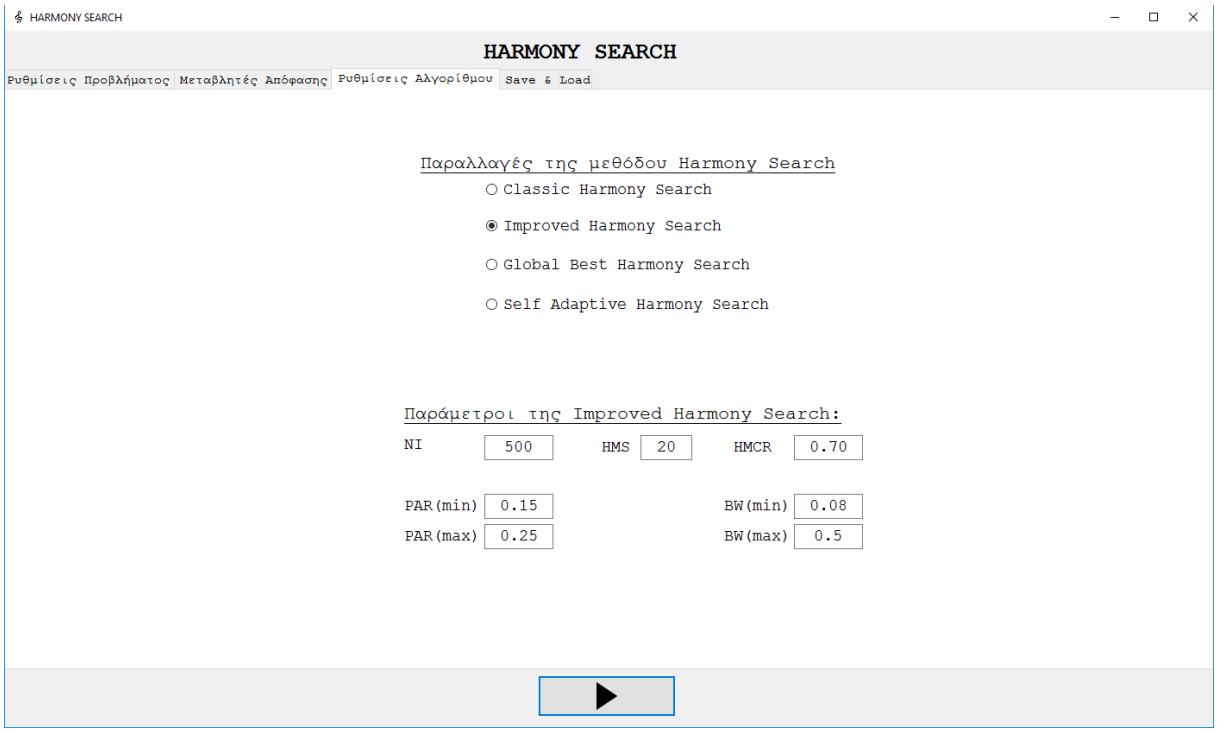
ΙΧ. ΟΔΗΓΙΕΣ ΧΡΗΣΗΣ



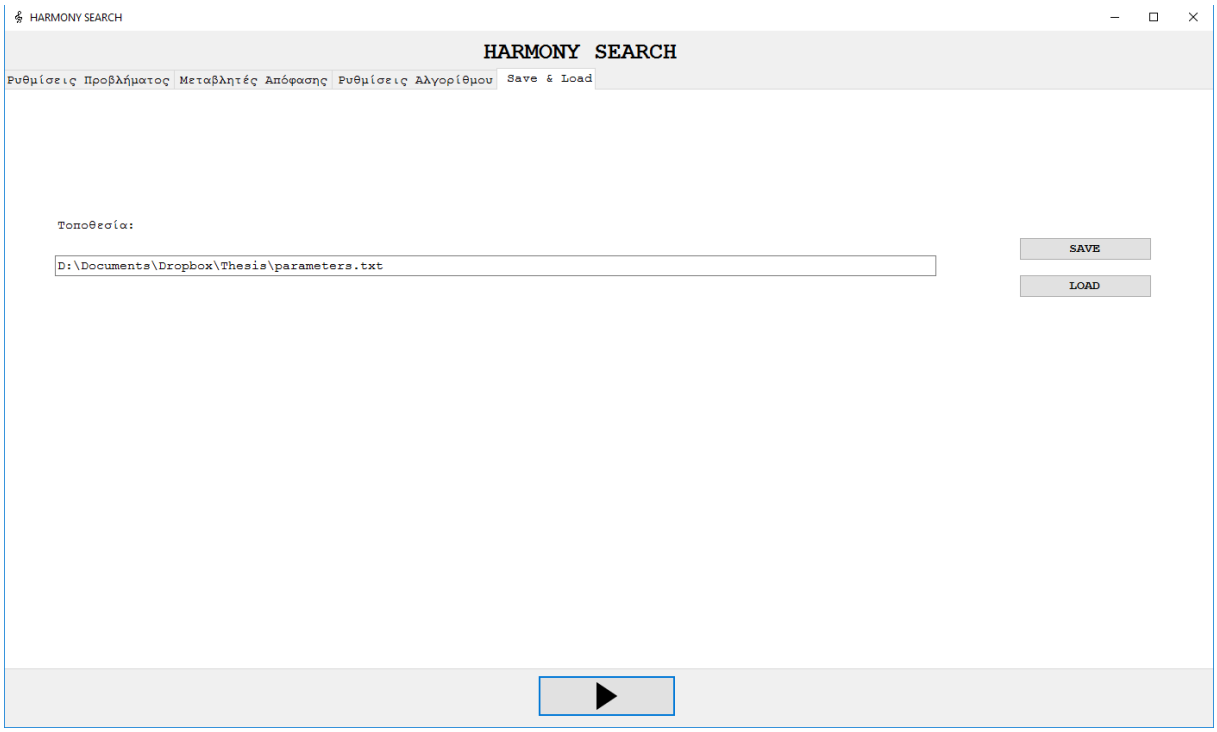
Η παραπάνω εικόνα είναι η πρώτη που εμφανίζει το πρόγραμμα όταν ξεκινήσει. Παρατηρείται ότι υπάρχουν 3 καρτέλες. Η πρώτη καρτέλα έχει τον τίτλο “Problem Settings”. Σε αυτή την καρτέλα καθορίζεται η αντικειμενική συνάρτηση. Ο χρήστης μπορεί να πληκτρολογήσει οποιαδήποτε μαθηματική έκφραση θέλει να βελτιστοποιήσει. Οι κανόνες των μαθηματικών εκφράσεων περιγράφονται στον σύνδεσμο που υπάρχει στο κάτω μέρος της οθόνης με μπλε γράμματα. Επιπλέον, ιδιαίτερη σημασία πρέπει να δοθεί στις μεταβλητές απόφασης. Όλες οι μεταβλητές απόφασης θα πρέπει να γράφονται με την μορφή x1, x2, … xn. Διαφορετικά, αν δηλαδή οι μεταβλητές απόφασης δεν γραφούν με αυτήν την μορφή, το πρόγραμμα θα δεν θα εκτελεστεί. Κάτω από την αντικειμενική συνάρτηση υπάρχουν οι επιλογές Maximize και Minimize. Η επιλογή αυτή καθορίζει αν η αντικειμενική συνάρτηση πρόκειται να ελαχιστοποιηθεί ή να μεγιστοποιηθεί. Λίγο παρακάτω παρατηρείται ένα checkbox και δίπλα του μια ετικέτα που γράφει “Show all improvisations”. Όταν τσεκάρεται, τότε το πρόγραμμα τυπώνει όλες της αρμονίες όλων των επαναλήψεων σε ένα αρχείο κειμένου με το όνομα “Results.txt”. Η επιλογή αυτή καθιστά το πρόγραμμα αρκετά πιο αργό. Τέλος, έξω από την καρτέλα στο κάτω μέρος της φόρμας υπάρχει το κουμπί Play. Με αριστερό κλικ σε αυτό το πλήκτρο η εφαρμογή οδηγείται στην επόμενη φόρμα με τα αποτελέσματα του αλγορίθμου. Το συγκεκριμένο κουμπί βρίσκεται συνεχώς στο ίδιο σημείο της φόρμας ανεξάρτητα από την καρτέλα στην οποία βρίσκεται ο χρήστης.



Στην παραπάνω εικόνα φαίνεται στην δεύτερη καρτέλα η οποία φέρει τον τίτλο “Decision Variables”. Σε αυτή την καρτέλα καθορίζεται το πεδίο ορισμού όλων των μεταβλητών απόφασης ξεχωριστά. Τα πεδία δημιουργούνται δυναμικά και ανάλογα με το πλήθος των μεταβλητών απόφασης που περιέχονται στην αντικειμενική συνάρτηση. Πρέπει οπωσδήποτε να συμπληρωθούν, διαφορετικά ο αλγόριθμος δεν θα μπορέσει να εκτελεστεί. Ταυτόχρονα, θα πρέπει τα πεδία στα αριστερά(ολικά ελάχιστα) να έχουν μικρότερη τιμή από το αντίστοιχο πεδίο στα δεξιά τους(ολικά μέγιστα).

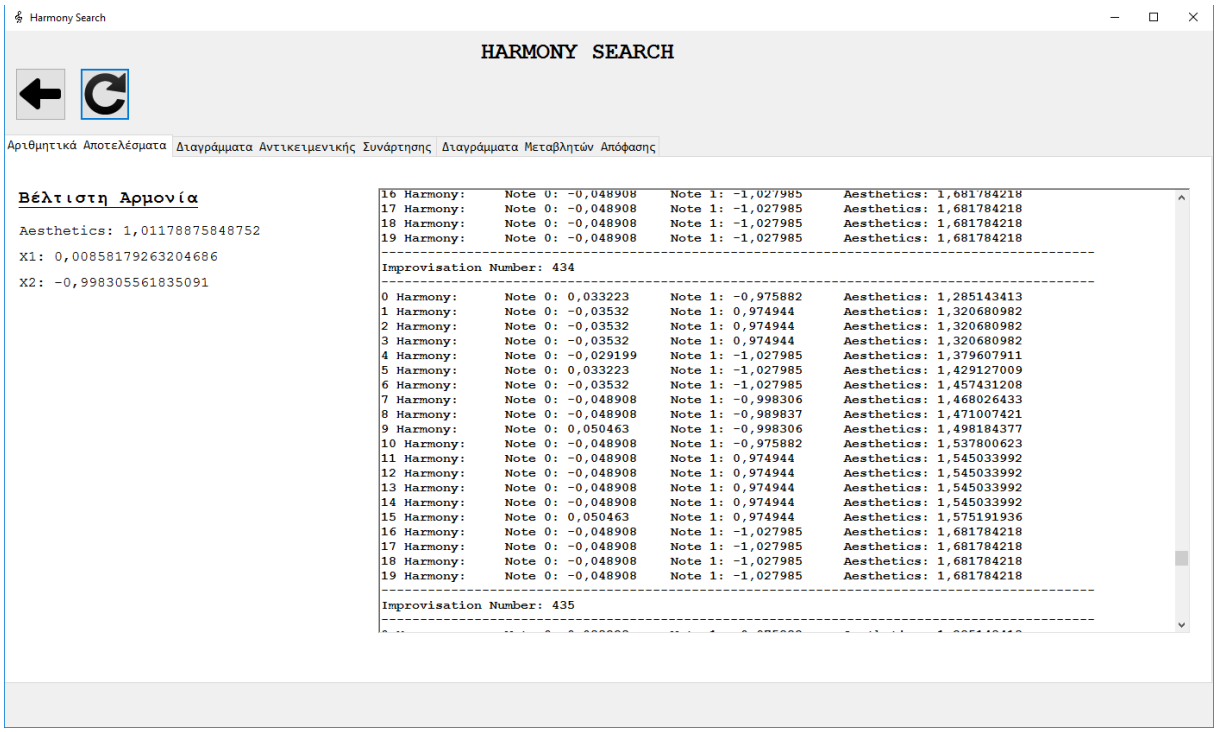


Η τρίτη καρτέλα έχει το όνομα “Algorithm Settings”. Σε αυτή την καρτέλα γίνονται ρυθμίσεις που αφορούν τον αλγόριθμο και όχι το πρόβλημα όπως γίνεται στις προηγούμενες δύο καρτέλες. Συγκεκριμένα, ο χρήστης μπορεί να επιλέξει ανάμεσα στις 4 βασικές παραλλαγές του αλγορίθμου Harmony Search. Κάθε φορά που ο χρήστης πλοηγείται ανάμεσα σε κάποια από τις παραλλαγές τότε εμφανίζονται και τα πεδία των παραμέτρων που αντιστοιχούν στην εκάστοτε παραλλαγή. Ο χρήστης μπορεί να εισάγει σε κάθε πεδίο όποια τιμή επιθυμεί. Αυτή είναι και η βασική καρτέλα στην οποία γίνονται πειραματισμοί έως ότου βρεθεί ένας ικανοποιητικός συνδυασμός παραμέτρων που θα κάνει τον αλγόριθμο να παράγει ικανοποιητικά αποτελέσματα.



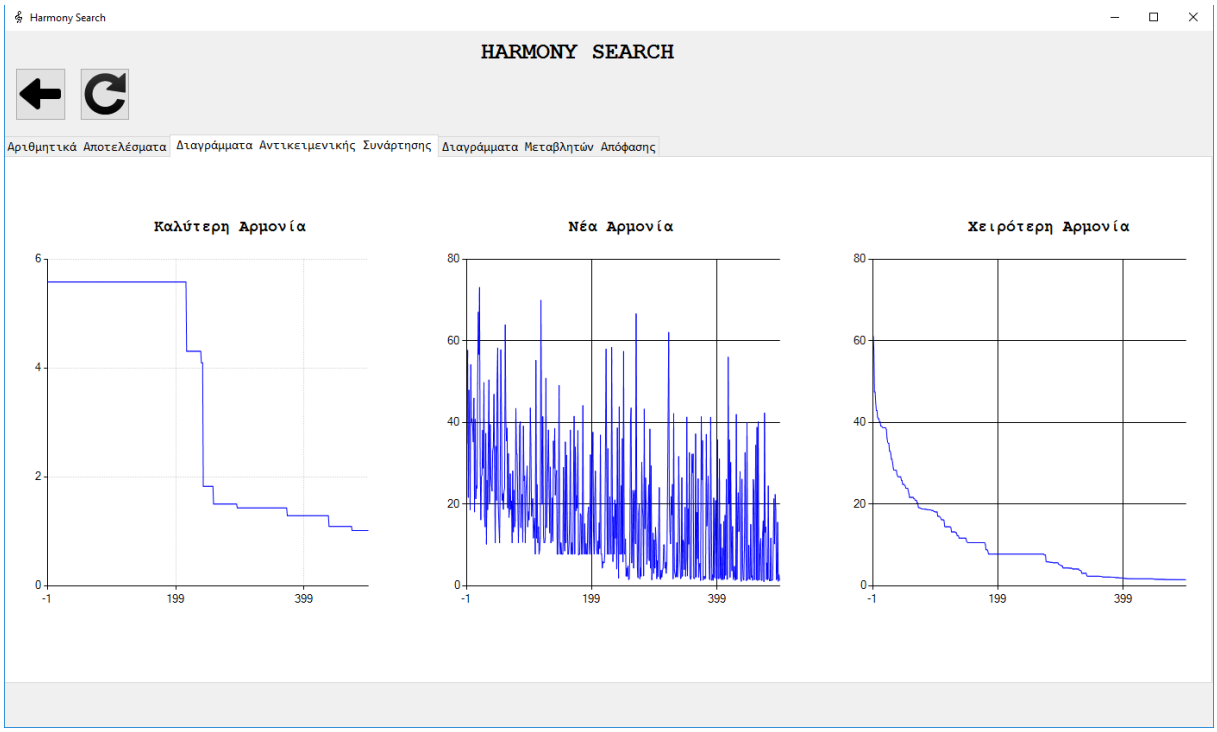
Εφόσον ο χρήστης έχει συμπληρώσει όλα τα πεδία και έχει εξακριβώσει ότι εισήχθησαν επιτρεπτές τιμές τότε μπορεί να κάνει αριστερό κλικ στο κουμπί play ώστε να δει και να μελετήσει τα αποτελέσματα του αλγορίθμου. Παράλληλα, υπάρχει μια ακόμα καρτέλα("Save & Load") στην οποία ο χρήστης μπορεί να αποθηκεύσει τις παραμέτρους που έδωσε στον αλγόριθμο για να τις χρησιμοποιήσει μια άλλη φορά.

Η δεύτερη φόρμα παρουσιάζει τα αποτελέσματα που παρήγαγε ο αλγόριθμος στο πρόβλημα που καλείται να αντιμετωπίσει. Οι τιμές που έλαβαν οι παράμετροί του είναι καθοριστικής σημασίας για την αποτελεσματικότητα του.

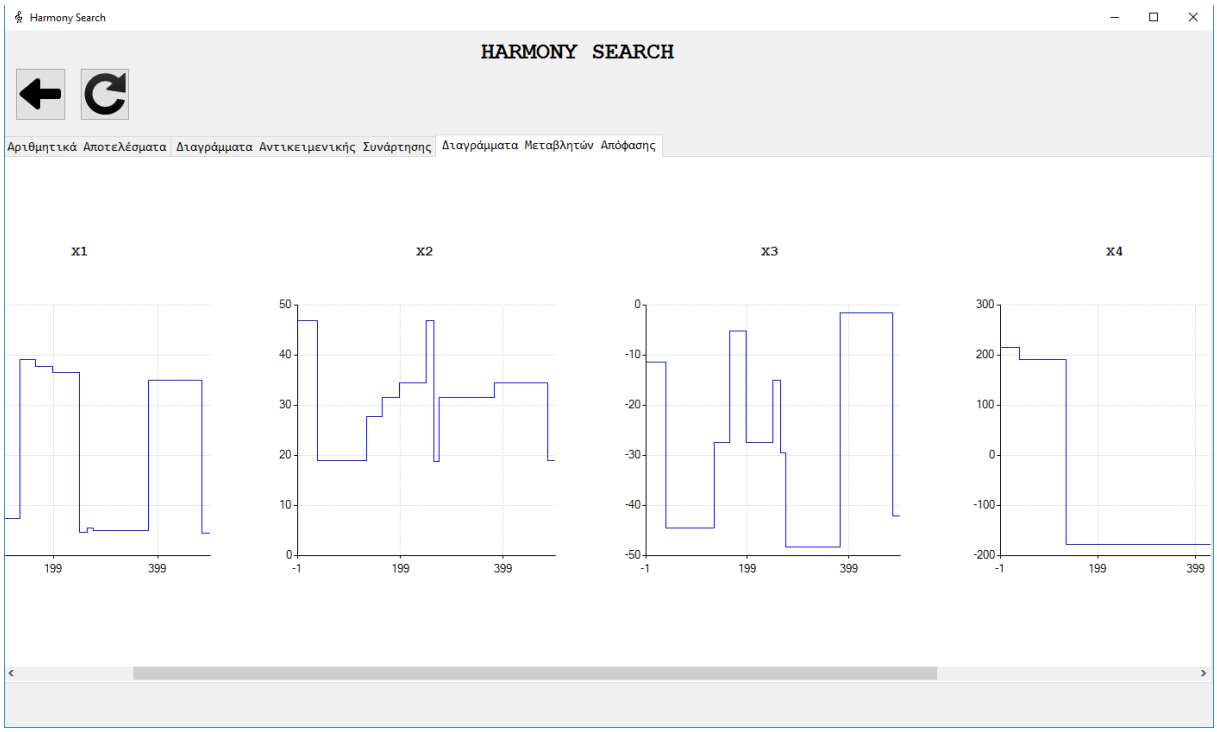


Αρχικά, πάνω αριστερά και έξω από τις καρτέλες παρατηρούνται δύο κουμπιά. Το κουμπί που βρίσκεται αριστερά οδηγεί την εφαρμογή στην προηγούμενη φόρμα, η οποία θα βρίσκεται στην μορφή που ήταν την τελευταία φορά που την έβλεπε ο χρήστης. Το δεύτερο κουμπί που βρίσκεται ακριβώς δεξιά του πρώτου, όταν ενεργοποιηθεί τρέχει τον αλγόριθμο ξανά από την αρχή χρησιμοποιώντας τις ρυθμίσεις που έχει ορίσει ο χρήστης στην προηγούμενη φόρμα. Η λειτουργία αυτή είναι πολύ σημαντική γιατί η μέθοδος Harmony Search είναι μια στοχαστική διαδικασία που σημαίνει ότι μπορεί να μην είναι εξίσου αποδοτική σε κάθε εκτέλεση.

Στην πρώτη καρτέλα “Αριθμητικά Αποτελέσματα” περιγράφονται τα αποτελέσματα της αναζήτησης με αριθμητικό και ακριβή τρόπο. Ενώ υπάρχει και ένας μεγάλος χώρος όπου φαίνονται τα αποτελέσματα της αναζήτησης σε κάθε γενιά.



Η δεύτερη καρτέλα φέρει τον τίτλο “Διαγράμματα Αντικειμενικής Συνάρτησης”. Εδώ, δίνεται η ευκαιρία στον χρήστη να παρατηρήσει τα αποτελέσματα του αλγορίθμου και την εξέλιξή που είχαν μέσα στο χρόνο. Τα γραφήματα αυτά είναι χρήσιμα εργαλεία για την εξαγωγή συμπερασμάτων που μπορεί να οδηγήσουν στον βέλτιστο ορισμό των παραμέτρων του αλγορίθμου. Στην πρώτη γραφική παράσταση, περιγράφεται η καλύτερη αρμονία της μνήμης της κάθε επανάληψης και η “αισθητική” που αυτή έχει. Είναι εύκολο να παρατηρηθεί αν ο αλγόριθμος κόλλησε ή αν παρέκκλινε στιγμιαία σε κάποιο τοπικό ακρότατο. Στο δεύτερο διάγραμμα, αποτυπώνεται η αισθητική της κάθε νέας αρμονίας που δημιουργείται. Από αυτό το διάγραμμα μπορεί να παρατηρηθεί αν οι μηχανισμοί δημιουργίας νέας αρμονίας είναι αποδοτικοί ή όχι, και κατά πόσο παρεκκλίνουν από τις αρμονίες της μνήμης. Το τρίτο διάγραμμα αφορά τη χειρότερη αρμονία της μνήμης της κάθε επανάληψης σε όρους αισθητικής. Σε αυτό το διάγραμμα μπορεί να παρατηρηθεί η συνολική εξέλιξη της μνήμης μέσα από πάροδο του χρόνου.



Στην τρίτη καρτέλα με τίτλο ‘Διαγράμματα Μεταβλητών Απόφασης’ αποτυπώνονται οι τιμές των μεταβλητών απόφασης μέσα από την πάροδο των επαναλήψεων. Έτσι, δίνεται η ευκαιρία να παρατηρηθεί αν κάποια μεταβλητή απόφασης εγκλωβίζεται ή αν βελτιστοποιείται με το πέρασμα των επαναλήψεων.

Χ. ΣΥΜΠΕΡΑΣΜΑΤΑ

Στην παρούσα εργασία ερευνήθηκε και αναπτύχθηκε ο αλγόριθμος Αναζήτησης Αρμονίας. Η έρευνα ξεκίνησε από την μελέτη των θεμελίων και των βασικών αρχών της βελτιστοποίησης και κατέληξε στην εφαρμογή της Αναζήτησης Αρμονίας σε προβλήματα δοκιμών.

Η μελέτη των χαρακτηριστικών της βελτιστοποίησης και των ευριστικών μεθόδων ήταν εποικοδομητική και συντέλεσε στην κατανόηση των επιμέρους μηχανισμών της Αναζήτησης Αρμονίας.

Η Αναζήτηση Αρμονίας εφαρμόστηκε με επιτυχία σε 5 προβλήματα δοκιμών. Για κάθε ένα από αυτά τα προβλήματα, ο αλγόριθμος κατάφερε να προσεγγίσει το ολικό βέλτιστο με ακρίβεια αρκετών δεκαδικών ψηφίων. Ωστόσο, παρατηρήθηκε ότι η απόδοση της μεθόδου εξαρτάται σε μεγάλο βαθμό από την ρύθμιση των παραμέτρων της. Υπήρχαν συνδυασμοί παραμέτρων που έδιναν καλά αποτελέσματα και το αντίστροφο. Για αυτόν τον λόγο, στις μετρήσεις που παρουσιάστηκαν σε προηγούμενο κεφάλαιο έγινε χρήση πολλών διαφορετικών συνδυασμών παραμέτρων. Φυσικά, μεγάλη ήταν και η σημασία του προβλήματος που αντιμετωπίζονταν κάθε φορά. Δεν είχαν όλοι οι συνδυασμοί παραμέτρων την ίδια απόδοση για κάθε πρόβλημα.

Το πρόγραμμα που δημιουργήθηκε για τις ανάγκες της παρούσας εργασίας μπορεί εύκολα να δοκιμαστεί σε οποιαδήποτε μαθηματική συνάρτηση εισάγει ο χρήστης και για οποιονδήποτε συνδυασμό παραμέτρων.

XI. ΒΙΒΛΙΟΓΡΑΦΙΑ

1. Sorensen, K., Sevaux, M., Glover, F., "A History of Metaheuristics", (2016), pp. 1-13

2. Κούγιας, Ι.Π., "Ο Αλγόριθμος Αναζήτησης Αρμονίας σε εφαρμογές Διαχείρισης Υδάτινων Πόρων", (2012)

3. Kiranyaz, S., Ince, T., Gabbouj, M., "Optimization Techniques: An Overview", (2014), pp. 13-29

4. Theodossiou, N., Kougias. I., Kakoudakis, K., Doikos, K., "Harmony Search Algorithm, a novel optimization technique. Parameter calibration and applications on water resources", 2011, pp. 245-249

5. Alia, O., Mandava, R., "The variants of the harmony search algorithm: an overview", (2011)

6. Wang, X., "The Overview of harmony Search", (2015), pp. 5-10

7. Wang, C.M., Huang, Y.F, "Seld-adaptive harmony search algorithm for optimization", (2009), pp. 2827-2836

8. Ashrafi, S.M., Dariane, A.B., "Performance evaluation of an improved harmony seach algorithm for numerical optimization: Melody Search(MS)", (2012), pp. 1301-1321

9. Geem, Z.W., Sim, K.B., "Parameter-setting-free harmony search algorithm", (2010), pp. 3881-3889

10. Mukhopadhayay, A., Roy, A., Das, S., Das, S., Abraham, A., "Population-Variance and Expolrative Power of Harmony Search: An Analysis"

11. Geem, Z.W., "Improved Harmony Search from Ensemble of Music Players", (2006), pp. 86-93

12. Omran, M., Mahdavi, M., "Global-best Harmony Search", (2008), pp. 1-15

13. Yang, X.S., "Harmony Search as a Metaheuristic Algorithm", (2009), pp. 1-17

14. Mahdavi, M., Fesanghary, M., Damangir, E., "An improved harmony search algorithm for solving optimization problems", (2007), pp. 1567-1579

15. Galip, E., Galip, F., "A Performance Study for Harmony Search Algorithm", (2015), pp. 31-35

16. https://en.wikipedia.org/wiki/No_free_lunch_in_search_and_optimization

17. http://gizmodo.com/scientists-are-turning-their-backs-on-algorithms-inspir-1708656813

18. https://en.wikipedia.org/wiki/Music_and_mathematics

19. http://www.musicheaven.gr/html/modules.php?name=News&file=article&id=432

20. https://blog.thinknewfound.com/2013/01/understanding-optimization-a-taxonomy-of-optimization-algorithms/

ΠΑΡΑΡΤΗΜΑ

ClassicSearch.cs

```
public class ClassicSearch : Search
{
        public float HMCR { get; set; }
        public float PAR { get; set; }
        public double BW { get; set; }

        private static ClassicSearch instance = null;

        private ClassicSearch()
        {
            initializeMemory();
        }

        public static ClassicSearch Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new ClassicSearch();
                }
                return instance;
            }
        }


        public void initializeMemory()
        {
            Results = new StringBuilder();
            base.bestHarmoniesAesthetics = new double[NI];
            base.newHarmoniesAesthetics = new double[NI];
            base.worstHarmoniesAesthetics = new double[NI];
            base.bestHarmoniesNotes = new double[NI, TotalNotes];
            base.Memory = new Harmony[HMSize];
            for (int i = 0; i < HMSize; i++)
            {
                Memory[i] = getRandomHarmony();
            }
            sortMemory();
        }

        public void Run(ProgressBar progressBar)
        {
              for (int currentImprovisation = 0; currentImprovisation < NI;
currentImprovisation++)
            {
                Harmony newHarmony = new Harmony();
                newHarmony.notes = new double[TotalNotes];
```

```
                    for (int currentNote = 0; currentNote < TotalNotes;
currentNote++)
                {
                        float randomFloat = Randomizer.getRandomFloat(0.0f,
1.0f);
                    if (randomFloat <= HMCR)
                    {
                        //Debug.WriteLine("HMCR");
                                                        int randomHarmony =
Convert.ToInt32(Randomizer.getRandomDouble(0, HMSize - 1));
                                                    newHarmony.notes[currentNote] =
base.Memory[randomHarmony].notes[currentNote];
                        adjustPitch(newHarmony, currentNote);
                    }
                    else
                    {
                                                    newHarmony.notes[currentNote] =
Randomizer.getRandomDouble(MinimumValues[currentNote],
MaximumValues[currentNote]);
                    }
                }
                base.updateMemory(newHarmony, currentImprovisation);
                        base.bestHarmoniesAesthetics[currentImprovisation] =
getHarmonyAesthetics(Memory[0]);
                        base.newHarmoniesAesthetics[currentImprovisation] =
getHarmonyAesthetics(newHarmony);
                    base.worstHarmoniesAesthetics[currentImprovisation] =
getHarmonyAesthetics(Memory[HMSize - 1]);
                for (int i = 0; i < TotalNotes; i++)
                {
                         base.bestHarmoniesNotes[currentImprovisation, i] =
Memory[0].notes.ElementAt(i);
                }

                if (ShowAll == true)
                {
                    base.writeResults(currentImprovisation);
                }

                progressBar.Value = currentImprovisation;
                        int percent = (int)(((double)(progressBar.Value -
progressBar.Minimum) / (double)(progressBar.Maximum - progressBar.Minimum))
* 100);
                 string progressMessage = "Please wait... Harmony Search in
progress. " + percent.ToString() + "%" + " completed.";
                                ControlStyle.SetProgressBarText(progressBar,
progressMessage,             ControlStyle.ProgressBarTextLocation.Centered,
Color.Black, new Font("Arial", 16));
            }
        }

        private void adjustPitch(Harmony newHarmony, int index)
        {
            float randomFloat = Randomizer.getRandomFloat(0.0f, 1.0f);
            if (randomFloat <= PAR)
            {
                randomFloat = Randomizer.getRandomFloat(-10.0f, 10.0f);
                if (randomFloat < 0)
```

```
                                                    newHarmony.notes[index]   +=
Randomizer.getRandomDouble(0, BW);
                else if (randomFloat >= 0)
                                                    newHarmony.notes[index]   -=
Randomizer.getRandomDouble(0, BW);

                                                     newHarmony.notes[index]    =
restrictNote(newHarmony.notes[index], index);
            }
        }

    }
```

ConfigurationForm.cs

```
public partial class ConfigurationForm : Form
    {
        private ClassicSearch classicHS;
        private ImprovedSearch improvedHS;
        private GlobalBestSearch globalHS;
        private SelfAdaptiveSearch adaptiveHS;

        private Expression objective;
        private HarmonySearchVariant currentVariant;

        private int totalNotesControls = 2;

        public ConfigurationForm()
        {
            InitializeComponent();
            ObjectiveRichTextBox.SelectAll();
            ObjectiveRichTextBox.SelectionAlignment = HorizontalAlignment.Center;
            ObjectiveRichTextBox.DeselectAll();
        }

        private void playButton_Click(object sender, EventArgs e)
        {
            if (isInputOk())
                setHarmonySearch();
            else
                return;

            GraphsForm graphs = new GraphsForm();
            graphs.classicHS = classicHS;
            graphs.improvedHS = improvedHS;
            graphs.globalHS = globalHS;
            graphs.adaptiveHS = adaptiveHS;
            graphs.ShowAll = showAllCheckBox.Checked;
            graphs.activationFlag = true;
            graphs.Configuration = this;

            this.Hide();
            graphs.Show();
        }

        private void setHarmonySearch()
        {
            if (currentVariant == HarmonySearchVariant.Classic)
            {
```

```
                classicHS = ClassicSearch.Instance;
                classicHS.Objective = objective;
                if (MaxRadioBtn.Checked)
                    classicHS.Optimum = OptimizationGoal.Max;
                if (MinRadioBtn.Checked)
                    classicHS.Optimum = OptimizationGoal.Min;
                if (MinAbsRadioBtn.Checked)
                    classicHS.Optimum = OptimizationGoal.MinAbs;
                classicHS.NI = Convert.ToInt32(NITextBox.Text);
                classicHS.TotalNotes = countDecisionVariables();
                classicHS.MinimumValues = new double[classicHS.TotalNotes];
                classicHS.MaximumValues = new double[classicHS.TotalNotes];
                for (int i = 0; i < classicHS.TotalNotes; i++)
                {
                    TextBox minTextBox = (TextBox)this.Controls.Find("x" + (i + 1) +
"MinTextBox", true)[0];
                        classicHS.MinimumValues[i] = double.Parse(minTextBox.Text,
CultureInfo.InvariantCulture);
                    TextBox maxTextBox = (TextBox)this.Controls.Find("x" + (i + 1) +
"MaxTextBox", true)[0];
                        classicHS.MaximumValues[i] = double.Parse(maxTextBox.Text,
CultureInfo.InvariantCulture);
                }
                classicHS.HMSize = Convert.ToInt32(HMSTextBox.Text);
                                    classicHS.HMCR   =  float.Parse(HMCRTextBox.Text,
CultureInfo.InvariantCulture);
                                      classicHS.PAR   =  float.Parse(PARTextBox.Text,
CultureInfo.InvariantCulture);
                                       classicHS.BW   =  double.Parse(BWTextBox.Text,
CultureInfo.InvariantCulture);
            }
            if (currentVariant == HarmonySearchVariant.Improved)
            {
                improvedHS = ImprovedSearch.Instance;
                improvedHS.Objective = objective;
                if (MaxRadioBtn.Checked)
                    improvedHS.Optimum = OptimizationGoal.Max;
                if (MinRadioBtn.Checked)
                    improvedHS.Optimum = OptimizationGoal.Min;
                if (MinAbsRadioBtn.Checked)
                    improvedHS.Optimum = OptimizationGoal.MinAbs;
                improvedHS.NI = Convert.ToInt32(NITextBox.Text);
                improvedHS.TotalNotes = countDecisionVariables();
                improvedHS.MinimumValues = new double[improvedHS.TotalNotes];
                improvedHS.MaximumValues = new double[improvedHS.TotalNotes];
                for (int i = 0; i < improvedHS.TotalNotes; i++)
                {
                    TextBox minTextBox = (TextBox)this.Controls.Find("x" + (i + 1) +
"MinTextBox", true)[0];
                        improvedHS.MinimumValues[i] = double.Parse(minTextBox.Text,
CultureInfo.InvariantCulture);
                    TextBox maxTextBox = (TextBox)this.Controls.Find("x" + (i + 1) +
"MaxTextBox", true)[0];
                        improvedHS.MaximumValues[i] = double.Parse(maxTextBox.Text,
CultureInfo.InvariantCulture);
                }
                improvedHS.HMSize = Convert.ToInt32(HMSTextBox.Text);
                                    improvedHS.HMCR   =  float.Parse(HMCRTextBox.Text,
CultureInfo.InvariantCulture);
                                  improvedHS.PARmin   =  float.Parse(PARMinTextBox.Text,
CultureInfo.InvariantCulture);
                                  improvedHS.PARmax   =  float.Parse(PARMaxTextBox.Text,
CultureInfo.InvariantCulture);
                                    improvedHS.BWmin   =  float.Parse(BWMinTextBox.Text,
CultureInfo.InvariantCulture);
```

```
                                    improvedHS.BWmax   =   float.Parse(BWMaxTextBox.Text,
CultureInfo.InvariantCulture);
            }
            if (currentVariant == HarmonySearchVariant.GlobalBest)
            {
                globalHS = GlobalBestSearch.Instance;
                globalHS.Objective = objective;
                if (MaxRadioBtn.Checked)
                    globalHS.Optimum = OptimizationGoal.Max;
                if (MinRadioBtn.Checked)
                    globalHS.Optimum = OptimizationGoal.Min;
                if (MinAbsRadioBtn.Checked)
                    globalHS.Optimum = OptimizationGoal.MinAbs;
                globalHS.NI = Convert.ToInt32(NITextBox.Text);
                globalHS.TotalNotes = countDecisionVariables();
                globalHS.MinimumValues = new double[globalHS.TotalNotes];
                globalHS.MaximumValues = new double[globalHS.TotalNotes];
                for (int i = 0; i < globalHS.TotalNotes; i++)
                {
                    TextBox minTextBox = (TextBox)this.Controls.Find("x" + (i + 1) +
"MinTextBox", true)[0];
                            globalHS.MinimumValues[i] = double.Parse(minTextBox.Text,
CultureInfo.InvariantCulture);
                    TextBox maxTextBox = (TextBox)this.Controls.Find("x" + (i + 1) +
"MaxTextBox", true)[0];
                            globalHS.MaximumValues[i] = double.Parse(maxTextBox.Text,
CultureInfo.InvariantCulture);
                }
                globalHS.HMSize = Convert.ToInt32(HMSTextBox.Text);
                                    globalHS.HMCR   =   float.Parse(HMCRTextBox.Text,
CultureInfo.InvariantCulture);
                                      globalHS.PAR   =   float.Parse(PARTextBox.Text,
CultureInfo.InvariantCulture);
            }
            if (currentVariant == HarmonySearchVariant.SelfAdaptive)
            {
                adaptiveHS = SelfAdaptiveSearch.Instance;
                adaptiveHS.Objective = objective;
                if (MaxRadioBtn.Checked)
                    adaptiveHS.Optimum = OptimizationGoal.Max;
                if (MinRadioBtn.Checked)
                    adaptiveHS.Optimum = OptimizationGoal.Min;
                if (MinAbsRadioBtn.Checked)
                    adaptiveHS.Optimum = OptimizationGoal.MinAbs;
                adaptiveHS.NI = Convert.ToInt32(NITextBox.Text);
                adaptiveHS.TotalNotes = countDecisionVariables();
                adaptiveHS.MinimumValues = new double[adaptiveHS.TotalNotes];
                adaptiveHS.MaximumValues = new double[adaptiveHS.TotalNotes];
                for (int i = 0; i < adaptiveHS.TotalNotes; i++)
                {
                    TextBox minTextBox = (TextBox)this.Controls.Find("x" + (i + 1) +
"MinTextBox", true)[0];
                        adaptiveHS.MinimumValues[i] = double.Parse(minTextBox.Text,
CultureInfo.InvariantCulture);
                    TextBox maxTextBox = (TextBox)this.Controls.Find("x" + (i + 1) +
"MaxTextBox", true)[0];
                        adaptiveHS.MaximumValues[i] = double.Parse(maxTextBox.Text,
CultureInfo.InvariantCulture);
                }
                adaptiveHS.HMSize = Convert.ToInt32(HMSTextBox.Text);
                                  adaptiveHS.HMCR   =   float.Parse(HMCRTextBox.Text,
CultureInfo.InvariantCulture);
                                    adaptiveHS.PAR   =   float.Parse(PARTextBox.Text,
CultureInfo.InvariantCulture);
            }
```

```
        }

        private Boolean isInputOk()
        {
              //if(ObjectiveRichTextBox.Text.Equals("") || ObjectiveRichTextBox.Text
== null)
            //{
              //     ControlStyle.MessageBoxStyle("The objective function is null or
empty.");
            //     return false;
            //}
            objective = new Expression(ObjectiveRichTextBox.Text);
            //if (objective.HasErrors())
            //{
            //     ControlStyle.MessageBoxStyle("The objective function is not valid.
\n\n" + objective.Error);
            //     return false;
            //}
            //if (objective == null)
            //{
            //     ControlStyle.MessageBoxStyle("The objective function is not valid.
Please try again.");
            //     return false;
            //}
            if (totalNotesControls < 2)
            {
                  //ControlStyle.MessageBoxStyle("The algorithm requires at least 2
decision variables.");
                   ControlStyle.MessageBoxStyle("Ο αλγόριθμος απαιτεί τουλάχιστον 2
μεταβλητές απόφασης.");
                return false;
            }
            try
            {
                for (int k=1; k <= totalNotesControls; k++)
                {
                    objective.Parameters["x" + k] = 0;
                }
                double res = (double)objective.Evaluate();
            }
            catch(Exception e)
            {
                //ControlStyle.MessageBoxStyle("The objective function is not valid.
Please try again.");
                        ControlStyle.MessageBoxStyle("Η αντικειμενική συνάρτηση δεν
συντάχθηκε σωστά.");
                return false;
            }

            for (int i = 0; i < totalNotesControls; i++)
            {
                  TextBox minTextBox = (TextBox)this.Controls.Find("x" + (i + 1) +
"MinTextBox", true)[0];
                  TextBox maxTextBox = (TextBox)this.Controls.Find("x" + (i + 1) +
"MaxTextBox", true)[0];
                    if (!ConfigurationRules.areExtremeValuesValid(minTextBox.Text,
maxTextBox.Text))
                {
                    //ControlStyle.MessageBoxStyle("Decision variable " + "X" + (i +
1) + " bounds are not valid.");
                     ControlStyle.MessageBoxStyle("Τα όρια της μεταβλητής απόφασης "
+ "X" + (i + 1) + " δεν είναι σωστά.");
                    return false;
                }
            }
```

```
            if (!ConfigurationRules.isNIValid(NITextBox.Text))
            {
                 //ControlStyle.MessageBoxStyle("NI(Number of Improvisations) field
is not valid. Please try again.");
                ControlStyle.MessageBoxStyle("Το πεδίο NI(Number of Improvisations)
δεν είναι σωστό.");
                return false;
            }
            if (!ConfigurationRules.isHMSValid(HMSTextBox.Text))
            {
                  //ControlStyle.MessageBoxStyle("HMS(Harmony Memory Size) field is
not valid. Please try again.");
                ControlStyle.MessageBoxStyle("Το πεδίο HMS(Harmony Memory Size) δεν
είναι σωστό.");
                return false;
            }
            if (!ConfigurationRules.isHMCRValid(HMCRTextBox.Text))
            {
                  //ControlStyle.MessageBoxStyle("HMCR(Harmony Memory Consideration
Rate) field is not valid. Please try again.");
                      ControlStyle.MessageBoxStyle("Το πεδίο HMCR(Harmony Memory
Consideration Rate) δεν είναι σωστό.");
                return false;
            }
            if (!currentVariant.Equals(HarmonySearchVariant.Improved))
            {
                if (!ConfigurationRules.isPARValid(PARTextBox.Text))
                {
                    //ControlStyle.MessageBoxStyle("PAR(Pitch Adjustment Rate) field
is not valid.");
                        ControlStyle.MessageBoxStyle("Το πεδίο PAR(Pitch Adjustment
Rate) δεν είναι σωστό.");
                    return false;
                }
                if(currentVariant.Equals(HarmonySearchVariant.Classic))
                {
                    if (!ConfigurationRules.isBWValid(BWTextBox.Text))
                    {
                          //ControlStyle.MessageBoxStyle("BW(Bandwidth) field is not
valid.");
                            ControlStyle.MessageBoxStyle("Το πεδίο BW(Bandwidth) δεν
είναι σωστό.");
                        return false;
                    }
                }
            }
            else
            {
                 if (!ConfigurationRules.arePARExtremesValid(PARMinTextBox.Text,
PARMaxTextBox.Text))
                {
                        //ControlStyle.MessageBoxStyle("PAR(Pitch Adjustment Rate)
bounds are not valid.");
                       ControlStyle.MessageBoxStyle("Τα όρια του πεδίου PAR(Pitch
Adjustment Rate) δεν είναι σωστά.");
                    return false;
                }
                   if (!ConfigurationRules.areBWExtremesValid(BWMinTextBox.Text,
BWMaxTextBox.Text))
                {
                    //ControlStyle.MessageBoxStyle("BW(Bandwidth) bounds are not not
valid.");
                     ControlStyle.MessageBoxStyle("Τα όρια του πεδίου BW(Bandwidth)
δεν είναι σωστά.");
                    return false;
```

```
                }
            }

            return true;
        }

        private void RadioButton_CheckedChanged(object sender, EventArgs e)
        {
            RadioButton senderRadioButton = sender as RadioButton;
            if(senderRadioButton.Tag.Equals("ClassicHS"))
            {
                PARTextBox.Visible = true;
                PARMinTextBox.Visible = false;
                PARMaxTextBox.Visible = false;
                BWTextBox.Visible = true;
                BWMaxTextBox.Visible = false;
                BWMinTextBox.Visible = false;
                PARLabel.Visible = true;
                PARMinLabel.Visible = false;
                PARMaxLabel.Visible = false;
                BWLabel.Visible = true;
                BWMaxLabel.Visible = false;
                BWMinLabel.Visible = false;
                      //ParametersLabel.Text = "Parameters of the Classic Harmony
Search:";
                ParametersLabel.Text = "Παράμετροι της Classic Harmony Search:";
                currentVariant = HarmonySearchVariant.Classic;
            }
            if (senderRadioButton.Tag.Equals("ImprovedHS"))
            {
                PARTextBox.Visible = false;
                PARMinTextBox.Visible = true;
                PARMaxTextBox.Visible = true;
                BWTextBox.Visible = false;
                BWMaxTextBox.Visible = true;
                BWMinTextBox.Visible = true;
                PARLabel.Visible = false;
                PARMinLabel.Visible = true;
                PARMaxLabel.Visible = true;
                BWLabel.Visible = false;
                BWMaxLabel.Visible = true;
                BWMinLabel.Visible = true;
                      //ParametersLabel.Text = "Parameters of the Improved Harmony
Search:";
                ParametersLabel.Text = "Παράμετροι της Improved Harmony Search:";
                currentVariant = HarmonySearchVariant.Improved;
            }
            if (senderRadioButton.Tag.Equals("GlobalBestHS"))
            {
                PARTextBox.Visible = true;
                PARMinTextBox.Visible = false;
                PARMaxTextBox.Visible = false;
                BWTextBox.Visible = false;
                BWMaxTextBox.Visible = false;
                BWMinTextBox.Visible = false;
                PARLabel.Visible = true;
                PARMinLabel.Visible = false;
                PARMaxLabel.Visible = false;
                BWLabel.Visible = false;
                BWMaxLabel.Visible = false;
                BWMinLabel.Visible = false;
                   //ParametersLabel.Text = "Parameters of the Global Best Harmony
Search:";
                ParametersLabel.Text = "Παράμετροι της Global Best Harmony Search:";
                currentVariant = HarmonySearchVariant.GlobalBest;
```

```
            }
            if (senderRadioButton.Tag.Equals("SelfAdaptiveHS"))
            {
                PARTextBox.Visible = true;
                PARMinTextBox.Visible = false;
                PARMaxTextBox.Visible = false;
                BWTextBox.Visible = false;
                BWMaxTextBox.Visible = false;
                BWMinTextBox.Visible = false;
                PARLabel.Visible = true;
                PARMinLabel.Visible = false;
                PARMaxLabel.Visible = false;
                BWLabel.Visible = false;
                BWMaxLabel.Visible = false;
                BWMinLabel.Visible = false;
                  //ParametersLabel.Text = "Parameters of the Self Adaptive Harmony
Search:";
                     ParametersLabel.Text = "Παράμετροι της Self Adaptive Harmony
Search:";
                currentVariant = HarmonySearchVariant.SelfAdaptive;
            }
        }

        private int countDecisionVariables()
        {
            int counter = 1;
            while(true)
            {
                if (ObjectiveRichTextBox.Text.Contains("x" + counter))
                    counter++;
                else
                    return --counter;
            }
        }

        private void createNewTextbox(int totalNotes)
        {
            for(int i = totalNotesControls + 1; i <= totalNotes; i++)
            {
                TextBox minTextBox = new TextBox();
                minTextBox.Name = "x" + i + "MinTextBox";
                TextBox maxTextBox = new TextBox();
                maxTextBox.Name = "x" + i + "MaxTextBox";
                Label label = new Label();
                label.Name = "x" + i + "Label";

                ControlStyle.TextBoxStyle(minTextBox, i);
                ControlStyle.TextBoxStyle(maxTextBox, i);
                ControlStyle.ConfigurationLabelStyle(label, i);
                variablesTab.Controls.Add(minTextBox);
                variablesTab.Controls.Add(maxTextBox);
                variablesTab.Controls.Add(label);
            }
        }

        private void deleteTextbox(int totalNotes)
        {
            for (int i = totalNotesControls; i > totalNotes; i--)
            {
                variablesTab.Controls.RemoveByKey("x" + i + "MinTextBox");
                variablesTab.Controls.RemoveByKey("x" + i + "MaxTextBox");
                variablesTab.Controls.RemoveByKey("x" + i + "Label");
            }
        }
```

```
        private void ConfigurationForm_FormClosed(object sender, FormClosedEventArgs
e)
        {
            Application.Exit();
        }

                            private    void    linkLabel1_LinkClicked(object    sender,
LinkLabelLinkClickedEventArgs e)
        {
            string url;
            if (e.Link.LinkData != null)
                url = e.Link.LinkData.ToString();
            else
                url = linkLabel1.Text.Substring(e.Link.Start, e.Link.Length);

            if (!url.Contains("://"))
                url = "http://" + url;

            var si = new ProcessStartInfo(url);
            Process.Start(si);
            linkLabel1.LinkVisited = true;
        }

        public void initializeComponents()
        {
            classicHS = null;
            improvedHS = null;
            globalHS = null;
            adaptiveHS = null;
        }

        private void ObjectiveRichTextBox_Enter(object sender, EventArgs e)
        {
            ObjectiveRichTextBox.SelectAll();
            ObjectiveRichTextBox.SelectionAlignment = HorizontalAlignment.Center;
            ObjectiveRichTextBox.DeselectAll();
        }

        private void ObjectiveRichTextBox_Leave(object sender, EventArgs e)
        {
            checkDecisionVariables();
        }

        private void checkDecisionVariables()
        {
            int totalNotes = countDecisionVariables();
            if (totalNotes > totalNotesControls)
            {
                createNewTextbox(totalNotes);
            }
            else if (totalNotes < totalNotesControls)
            {
                deleteTextbox(totalNotes);
            }
            totalNotesControls = totalNotes;
        }

        private void loadButton_Click(object sender, EventArgs e)
        {
            Parameters parameters = new Parameters();
            string path = pathTextBox.Text;
              if (String.IsNullOrEmpty(path) || String.IsNullOrWhiteSpace(path) || !
path.EndsWith(".txt") || !path.Contains("\\"))
            {
```

```
                ControlStyle.MessageBoxStyle("Η τοποθεσία(URI) του αρχείου δεν είναι
έγκυρη");
                return;
            }
            using (StreamReader r = new StreamReader(path))
            {
                string json = r.ReadToEnd();
                parameters = JsonConvert.DeserializeObject<Parameters>(json);
            }
            ObjectiveRichTextBox.Text = parameters.Objective;
            if (parameters.Optimum == OptimizationGoal.Max)
                MaxRadioBtn.Checked = true;
            if (parameters.Optimum == OptimizationGoal.Min)
                MinRadioBtn.Checked = true;
            if (parameters.Optimum == OptimizationGoal.MinAbs)
                MinAbsRadioBtn.Checked = true;
            showAllCheckBox.Checked = parameters.ShowAll;
            checkDecisionVariables();
            for (int j=0; j<totalNotesControls; j++)
            {
                    Controls.Find("x" + (j + 1) + "MinTextBox", true)[0].Text =
parameters.MinValues[j].ToString();
                    Controls.Find("x" + (j + 1) + "MaxTextBox", true)[0].Text =
parameters.MaxValues[j].ToString();
            }
            NITextBox.Text = parameters.NI.ToString();
            HMSTextBox.Text = parameters.HMS.ToString();
            HMCRLabel.Text = parameters.HMCR.ToString();
            PARTextBox.Text = parameters.PAR.ToString();
            BWTextBox.Text = parameters.BW.ToString();
            PARMinTextBox.Text = parameters.PARmin.ToString();
            PARMaxTextBox.Text = parameters.PARmax.ToString();
            BWMinTextBox.Text = parameters.PARmin.ToString();
            BWMaxTextBox.Text = parameters.PARmax.ToString();
            if (parameters.Variant == HarmonySearchVariant.Classic)
                ClassicRadioButton.Checked = true;
            if (parameters.Variant == HarmonySearchVariant.Improved)
                ImprovedRadioButton.Checked = true;
            if (parameters.Variant == HarmonySearchVariant.GlobalBest)
                GlobalRadioButton.Checked = true;
            if (parameters.Variant == HarmonySearchVariant.SelfAdaptive)
                AdaptiveRadioButton.Checked = true;

              ControlStyle.MessageBoxSuccessStyle("Οι παράμετροι του αλγορίθμου
φορτώθηκαν επιτυχώς από το αρχείο.");
        }

        private void saveButton_Click(object sender, EventArgs e)
        {
            if(!isInputOk())
            {
                //ControlStyle.MessageBoxStyle("Parameters are not valid.");
                ControlStyle.MessageBoxStyle("Οι παράμετροι δεν είναι σωστοί.");
                return;
            }

            string path = pathTextBox.Text;
             if (String.IsNullOrEmpty(path) || String.IsNullOrWhiteSpace(path) || !
path.EndsWith(".txt") || !path.Contains("\\"))
            {
                ControlStyle.MessageBoxStyle("Η τοποθεσία(URI) του αρχείου δεν είναι
έγκυρη");
                return;
            }
```

```
            Parameters parameters = new Parameters();

            parameters.Objective = ObjectiveRichTextBox.Text;
            parameters.ShowAll = showAllCheckBox.Checked;
            if (MaxRadioBtn.Checked == true)
                parameters.Optimum = OptimizationGoal.Max;
            if (MinRadioBtn.Checked == true)
                parameters.Optimum = OptimizationGoal.Min;
            if (MinAbsRadioBtn.Checked == true)
                parameters.Optimum = OptimizationGoal.MinAbs;
            parameters.MaxValues = new double[totalNotesControls];
            parameters.MinValues = new double[totalNotesControls];
            for (int i=0; i<totalNotesControls; i++)
            {
                  TextBox maxTextbox = (TextBox) this.Controls.Find("x" + (i + 1) +
"MaxTextBox", true)[0];
                        parameters.MaxValues[i] = double.Parse(maxTextbox.Text,
CultureInfo.InvariantCulture);
                  TextBox minTextbox = (TextBox) this.Controls.Find("x" + (i + 1) +
"MinTextBox", true)[0];
                        parameters.MinValues[i] = double.Parse(minTextbox.Text,
CultureInfo.InvariantCulture);
            }
            parameters.NI = Int32.Parse(NITextBox.Text);
            parameters.HMS = Int32.Parse(HMSTextBox.Text);
            parameters.HMCR = Convert.ToDouble(HMCRTextBox.Text);
            parameters.PAR = Convert.ToDouble(PARTextBox.Text);
            parameters.BW = Convert.ToDouble(BWTextBox.Text);
            parameters.PARmin = Convert.ToDouble(PARMinTextBox.Text);
            parameters.PARmax = Convert.ToDouble(PARMaxTextBox.Text);
            parameters.BWmin = Convert.ToDouble(BWMinTextBox.Text);
            parameters.BWmax = Convert.ToDouble(BWMaxTextBox.Text);
            if (ClassicRadioButton.Checked == true)
                parameters.Variant = HarmonySearchVariant.Classic;
            if (ImprovedRadioButton.Checked == true)
                parameters.Variant = HarmonySearchVariant.Improved;
            if (GlobalRadioButton.Checked == true)
                parameters.Variant = HarmonySearchVariant.GlobalBest;
            if (AdaptiveRadioButton.Checked == true)
                parameters.Variant = HarmonySearchVariant.SelfAdaptive;

            string json = JsonConvert.SerializeObject(parameters);
            File.WriteAllText(@path, json);

                  ControlStyle.MessageBoxSuccessStyle("Οι παράμετροι του αλγορίθμου
αποθηκεύτηκαν επιτυχώς σε αρχείο.");
        }
    }
```

ConfigurationRules.cs

```
public static class ConfigurationRules
    {
        public static Boolean isNIValid(String NIText)
        {
            if (NIText == null || NIText == "")
                return false;
            int NI = 0;
            if (!Int32.TryParse(NIText, out NI))
                return false;
```

```
            if (NI <= 0)
                return false;

            return true;
        }

        public static Boolean isHMSValid(String HMSText)
        {
            if (HMSText == null || HMSText == "")
                return false;
            int HMS = 0;
            if (!Int32.TryParse(HMSText, out HMS))
                return false;
            if (HMS <= 0)
                return false;

            return true;
        }

        public static Boolean areTotalVariablesValid(String TotalVarsText)
        {
            if (TotalVarsText == null || TotalVarsText == "")
                return false;
            int TotalVars = 0;
            if (!Int32.TryParse(TotalVarsText, out TotalVars))
                return false;
            if (TotalVars <= 0)
                return false;

            return true;
        }

        public static Boolean isHMCRValid(String HMCRText)
        {
            if (HMCRText == null || HMCRText == "")
                return false;
            float HMCR = 0.0f;
                                if   (!float.TryParse(HMCRText,   NumberStyles.Any,
CultureInfo.InvariantCulture, out HMCR))
                return false;
            if (HMCR <= 0)
                return false;
            if (HMCR > 1)
                return false;

            return true;
        }

        public static Boolean isPARValid(String PARText)
        {
            if (PARText == null || PARText == "")
                return false;
            float PAR = 0.0f;
                                  if   (!float.TryParse(PARText,   NumberStyles.Any,
CultureInfo.InvariantCulture, out PAR))
                return false;
            if (PAR <= 0)
                return false;
            if (PAR > 1)
                return false;

            return true;
        }
```

```
            public static Boolean arePARExtremesValid(String minPARText, String
maxPARText)
        {
             if (minPARText == null || minPARText == "" || maxPARText == null ||
maxPARText == "")
                return false;
            float minPAR = 0.0f;
                                if   (!float.TryParse(minPARText,   NumberStyles.Any,
CultureInfo.InvariantCulture, out minPAR))
                return false;
            if (minPAR <= 0)
                return false;
            if (minPAR >= 1)
                return false;
            float maxPAR = 0.0f;
                                if   (!float.TryParse(maxPARText,   NumberStyles.Any,
CultureInfo.InvariantCulture, out maxPAR))
                return false;
            if (maxPAR <= 0)
                return false;
            if (maxPAR >= 1)
                return false;
            if (minPAR >= maxPAR)
                return false;

            return true;
        }

        public static Boolean isBWValid(String BWText)
        {
            if (BWText == null || BWText == "")
                return false;
            double BW = 0.0;
                                  if   (!double.TryParse(BWText,   NumberStyles.Any,
CultureInfo.InvariantCulture, out BW))
                return false;

            return true;
        }

        public static Boolean areBWExtremesValid(String minBWText, String maxBWText)
        {
                if (minBWText == null || minBWText == "" || maxBWText == null ||
maxBWText == "")
                return false;
            float minBW = 0.0f;
                                if   (!float.TryParse(minBWText,   NumberStyles.Any,
CultureInfo.InvariantCulture, out minBW))
                return false;
            if (minBW <= 0)
                return false;
            float maxBW = 0.0f;
                                if   (!float.TryParse(maxBWText,   NumberStyles.Any,
CultureInfo.InvariantCulture, out maxBW))
                return false;
            if (maxBW <= 0)
                return false;
            if (minBW >= maxBW)
                return false;

            return true;
        }

           public static Boolean areExtremeValuesValid(String minValueText, String
maxValueText)
```

```
        {
             if (minValueText == null || maxValueText == null || minValueText == ""
|| maxValueText == "")
                 return false;
             double minValue = 0.0;
                             if   (!double.TryParse(minValueText,   NumberStyles.Any,
CultureInfo.InvariantCulture, out minValue))
                 return false;
             double maxValue = 0.0;
                             if   (!double.TryParse(maxValueText,   NumberStyles.Any,
CultureInfo.InvariantCulture, out maxValue))
                 return false;
             if (minValue >= maxValue)
                 return false;

             return true;
         }
     }
```

ControlStyle.cs

```
public static class ControlStyle
    {
           public static void SetProgressBarText(ProgressBar Target, string Text,
ProgressBarTextLocation Location, Color TextColor, Font TextFont)
         {
             if (Target == null)
             {
                 throw new ArgumentException("Null Target");
             }
             if (string.IsNullOrEmpty(Text))
             {
                     int percent = (int)(((double)(Target.Value - Target.Minimum) /
(double)(Target.Maximum - Target.Minimum)) * 100);
                 Text = percent.ToString() + "%";
             }
             using (Graphics gr = Target.CreateGraphics())
             {
                 gr.DrawString(Text, TextFont, new SolidBrush(TextColor),
                         new PointF(Location == ProgressBarTextLocation.Left ? 5 :
Target.Width / 2 - (gr.MeasureString(Text, TextFont).Width / 2.0F),
                             Target.Height / 2 - (gr.MeasureString(Text, TextFont).Height
/ 2.0F)));
             }
         }

         public enum ProgressBarTextLocation
         {
             Left, Centered
         }

         public static void MessageBoxStyle(String message)
         {
                 MessageBox.Show(message, "Harmony Search", MessageBoxButtons.OK,
MessageBoxIcon.Error);
         }

         public static void MessageBoxSuccessStyle(String message)
         {
```

```
                MessageBox.Show(message, "Harmony Search", MessageBoxButtons.OK,
MessageBoxIcon.Information);
        }

        public static void TextBoxStyle(TextBox textBox, int number)
        {
            textBox.Size = new Size(400, 22);
            textBox.TextAlign = HorizontalAlignment.Center;
            textBox.Font = new Font("Courier New", 10, FontStyle.Regular);
            if (textBox.Name.Contains("Min"))
            {
                textBox.Location = new Point(222, 100 + (50 * (number - 1)));
                textBox.Anchor = (AnchorStyles.Top | AnchorStyles.Left);
            }
            if(textBox.Name.Contains("Max"))
            {
                textBox.Location = new Point(888, 100 + (50 * (number - 1)));
                textBox.Anchor = (AnchorStyles.Top | AnchorStyles.Right);
            }
        }

        public static void ConfigurationLabelStyle(Label label, int number)
        {
            label.Text = "X" + number;
            label.Size = new Size(40, 20);
            label.ForeColor = Color.Black;
            label.Font = new Font("Courier New", 12, FontStyle.Bold);
            label.Visible = true;
            label.Location = new Point(150, 102 + (50 * (number - 1)));
        }

        public static void GraphsLabelStyle(Label label, int number)
        {
            label.Name = "x" + number + "Label";
            label.Text = "X" + number;
            label.Size = new Size(40, 20);
            label.ForeColor = Color.Black;
            label.Font = new Font("Courier New", 12, FontStyle.Bold);
            label.Visible = true;
            label.Location = new Point(275 + (400 * (number - 1)), 100);
        }

        public static void LabelStyle(Label label, double value)
        {
            label.Name = "aestheticsLabel";
            label.Text = "Aesthetics: " + value;
            label.Size = new Size(450, 18);
            label.ForeColor = Color.Black;
            label.Font = new Font("Courier New", 12, FontStyle.Regular);
            label.Visible = true;
            label.Location = new Point(15, 75);
        }

        public static void LabelStyle(Label label, int number, double value)
        {
            label.Name = "x" + number + "Label";
            label.Text = "X" + number + ": " + value;
            label.Size = new Size(308, 18);
            label.ForeColor = Color.Black;
            label.Font = new Font("Courier New", 12, FontStyle.Regular);
            label.Visible = true;
            label.Location = new Point(15, 105 + (30 * (number - 1)));
        }

        public static void ChartStyle(Chart chart, int number)
```

```
        {
            chart.Name = "x" + number + "Chart";
            chart.Size = new Size(350, 350);
            chart.Location = new Point(100 + (400 * (number -1)), 150);
            chart.Series.Clear();
            chart.Series.Add("X" + number + "Decision Variable");
            chart.Series[0].ChartType = SeriesChartType.StepLine;
            chart.Series[0].Color = System.Drawing.Color.Blue;
            chart.Visible = true;
            chart.ChartAreas.Add("X" + number + "Decision Variable");
                                    chart.ChartAreas[0].AxisX.MajorGrid.LineColor   =
System.Drawing.Color.LightGray;
            chart.ChartAreas[0].AxisX.MajorGrid.LineDashStyle = ChartDashStyle.Dot;
                                    chart.ChartAreas[0].AxisY.MajorGrid.LineColor   =
System.Drawing.Color.LightGray;
            chart.ChartAreas[0].AxisY.MajorGrid.LineDashStyle = ChartDashStyle.Dot;
        }
    }
```

GlobalBestSearch.cs

```
public class GlobalBestSearch : Search
    {
        public float HMCR { get; set; }
        public float PAR { get; set; }
        public double BW { get; set; }

        private static GlobalBestSearch instance = null;

        private GlobalBestSearch()
        {
            initializeMemory();
        }

        public static GlobalBestSearch Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new GlobalBestSearch();
                }
                return instance;
            }
        }

        public void initializeMemory()
        {
            Results = new StringBuilder();
            base.bestHarmoniesAesthetics = new double[NI];
            base.newHarmoniesAesthetics = new double[NI];
            base.worstHarmoniesAesthetics = new double[NI];
            base.bestHarmoniesNotes = new double[NI, TotalNotes];
            Memory = new Harmony[HMSize];
            for (int i = 0; i < HMSize; i++)
                Memory[i] = getRandomHarmony();

            sortMemory();
        }
```

```
        public void Run(ProgressBar progressBar)
        {
                    for (int currentImprovisation = 0; currentImprovisation < NI;
currentImprovisation++)
            {
                Harmony newHarmony = new Harmony();
                newHarmony.notes = new double[TotalNotes];
                for (int currentNote = 0; currentNote < TotalNotes; currentNote++)
                {
                    float randomFloat = Randomizer.getRandomFloat(0.0f, 1.0f);
                    if (randomFloat <= HMCR)
                    {
                                                                int randomHarmony =
Convert.ToInt32(Randomizer.getRandomDouble(0, HMSize - 1));
                                                            newHarmony.notes[currentNote] =
Memory[randomHarmony].notes[currentNote];
                        adjustPitch(newHarmony);
                    }
                    else
                    {
                                                            newHarmony.notes[currentNote] =
Randomizer.getRandomDouble(MinimumValues[currentNote], MaximumValues[currentNote]);
                    }
                }
                updateMemory(newHarmony, currentImprovisation);
                                base.bestHarmoniesAesthetics[currentImprovisation] =
getHarmonyAesthetics(Memory[0]);
                                 base.newHarmoniesAesthetics[currentImprovisation] =
getHarmonyAesthetics(newHarmony);
                                base.worstHarmoniesAesthetics[currentImprovisation] =
getHarmonyAesthetics(Memory[HMSize-1]);
                for (int i = 0; i < TotalNotes; i++)
                {
                                  base.bestHarmoniesNotes[currentImprovisation, i] =
Memory[0].notes.ElementAt(i);
                }
                if (ShowAll == true)
                {
                    base.writeResults(currentImprovisation);
                }

                progressBar.Value = currentImprovisation;
                                int percent = (int)(((double)(progressBar.Value -
progressBar.Minimum) / (double)(progressBar.Maximum - progressBar.Minimum)) * 100);
                string progressMessage = "Please wait... Harmony Search in progress.
" + percent.ToString() + "%" + " completed.";
                        ControlStyle.SetProgressBarText(progressBar, progressMessage,
ControlStyle.ProgressBarTextLocation.Centered, Color.Black, new Font("Arial", 16));
            }
        }

        private void adjustPitch(Harmony newHarmony)
        {
            float randomFloat = Randomizer.getRandomFloat(0.0f, 1.0f);
            int index = Randomizer.getRandomInteger(0, TotalNotes - 1);
            if (randomFloat <= PAR)
            {
                newHarmony.notes[index] = Memory[0].notes[index];
            }
            newHarmony.notes[index] = restrictNote(newHarmony.notes[index], index);
        }
}
```

GraphsForm.cs

```
public partial class GraphsForm : Form
    {
        public bool ShowAll { get; set; }
        private int totalNotes;
        public bool activationFlag;
        private bool isBackButtonPressed = false;

        public ClassicSearch classicHS { get; set; }
        public ImprovedSearch improvedHS { get; set; }
        public GlobalBestSearch globalHS { get; set; }
        public SelfAdaptiveSearch adaptiveHS { get; set; }

        public ConfigurationForm Configuration { get; set; }

        private List<double> results = new List<double>();

        public GraphsForm()
        {
            InitializeComponent();
        }

        private void GraphsForm_Activated(object sender, EventArgs e)
        {
            if (activationFlag == false)
                return;
            SearchProgress.Show();
            this.Enabled = false;
            performHarmonySearch();
            plotChart();
            SearchProgress.Hide();
            this.Enabled = true;
            activationFlag = false;
        }

        private void RerunButton_Click(object sender, EventArgs e)
        {
            SearchProgress.Show();
            this.Enabled = false;
            performHarmonySearch();
            numericsTab.Controls.RemoveByKey("aestheticsLabel");
            for (int k = 1; k <= totalNotes; k++)
            {
                numericsTab.Controls.RemoveByKey("x" + k + "Label");
                notesTab.Controls.RemoveByKey("x" + k + "Chart" );
            }
            plotChart();
            SearchProgress.Hide();
            this.Enabled = true;
        }

        private void plotChart()
        {
            BestHarmonyChart.Series.Clear();
            NewHarmonyChart.Series.Clear();
            WorstHarmonyChart.Series.Clear();
            //X1Chart.Series.Clear();
            //X2Chart.Series.Clear();
```

```
            if (classicHS != null)
            {
                if (classicHS.ShowAll == true)
                    resultsRichTextBox.Text = classicHS.Results.ToString();
                totalNotes = classicHS.TotalNotes;
                BestHarmonyChart.Series.Add("Aesthetics");
                BestHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
                BestHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                NewHarmonyChart.Series.Add("Aesthetics");
                NewHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
                NewHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                WorstHarmonyChart.Series.Add("Aesthetics");
                WorstHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
                WorstHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                List<Chart> allDecisionVariables = new List<Chart>();
                for (int i = 1; i <= classicHS.TotalNotes; i++)
                {
                    Chart noteChart = new Chart();
                    ControlStyle.ChartStyle(noteChart, i);
                    allDecisionVariables.Add(noteChart);
                    Label noteLabel = new Label();
                    ControlStyle.GraphsLabelStyle(noteLabel, i);
                    notesTab.Controls.Add(noteLabel);
                }
                for (int j = 0; j < classicHS.NI; j++)
                {
                                                BestHarmonyChart.Series[0].Points.AddXY(j,
classicHS.bestHarmoniesAesthetics[j]);
                                                 NewHarmonyChart.Series[0].Points.AddXY(j,
classicHS.newHarmoniesAesthetics[j]);
                                               WorstHarmonyChart.Series[0].Points.AddXY(j,
classicHS.worstHarmoniesAesthetics[j]);
                    for (int i = 1; i <= classicHS.TotalNotes; i++)
                    {
                                    allDecisionVariables[i-1].Series[0].Points.AddXY(j,
classicHS.bestHarmoniesNotes[j, i-1]);
                    }
                }
                for(int i = 0; i < classicHS.TotalNotes; i++)
                {
                    notesTab.Controls.Add(allDecisionVariables[i]);
                }
                Label aestheticsLabel = new Label();
                                                ControlStyle.LabelStyle(aestheticsLabel,
classicHS.bestHarmoniesAesthetics[classicHS.NI - 1]);
                numericsTab.Controls.Add(aestheticsLabel);
                for (int k = 1; k <= classicHS.TotalNotes; k++)
                {
                    Label note = new Label();
                                                    ControlStyle.LabelStyle(note,   k,
classicHS.bestHarmoniesNotes[classicHS.NI - 1, k - 1]);
                    numericsTab.Controls.Add(note);
                }

                saveResults(classicHS.bestHarmoniesAesthetics[classicHS.NI - 1]);
                //if (ShowAll == true)
                //    System.Diagnostics.Process.Start("Results.txt");
            }
            if (improvedHS != null)
            {
                if (improvedHS.ShowAll == true)
                    resultsRichTextBox.Text = improvedHS.Results.ToString();
                totalNotes = improvedHS.TotalNotes;
                BestHarmonyChart.Series.Add("Aesthetics");
                BestHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
```

```
                BestHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                NewHarmonyChart.Series.Add("Aesthetics");
                NewHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
                NewHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                WorstHarmonyChart.Series.Add("Aesthetics");
                WorstHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
                WorstHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                List<Chart> allDecisionVariables = new List<Chart>();
                for (int i = 1; i <= improvedHS.TotalNotes; i++)
                {
                    Chart noteChart = new Chart();
                    ControlStyle.ChartStyle(noteChart, i);
                    allDecisionVariables.Add(noteChart);
                    Label noteLabel = new Label();
                    ControlStyle.GraphsLabelStyle(noteLabel, i);
                    notesTab.Controls.Add(noteLabel);
                }
                for (int j = 0; j < improvedHS.NI; j++)
                {
                                            BestHarmonyChart.Series[0].Points.AddXY(j,
improvedHS.bestHarmoniesAesthetics[j]);
                                             NewHarmonyChart.Series[0].Points.AddXY(j,
improvedHS.newHarmoniesAesthetics[j]);
                                           WorstHarmonyChart.Series[0].Points.AddXY(j,
improvedHS.worstHarmoniesAesthetics[j]);
                    for (int i = 1; i <= improvedHS.TotalNotes; i++)
                    {
                                    allDecisionVariables[i - 1].Series[0].Points.AddXY(j,
improvedHS.bestHarmoniesNotes[j, i - 1]);
                    }
                }
                for (int i = 0; i < improvedHS.TotalNotes; i++)
                {
                    notesTab.Controls.Add(allDecisionVariables[i]);
                }
                Label aestheticsLabel = new Label();
                                              ControlStyle.LabelStyle(aestheticsLabel,
improvedHS.bestHarmoniesAesthetics[improvedHS.NI - 1]);
                numericsTab.Controls.Add(aestheticsLabel);
                for (int k = 1; k <= improvedHS.TotalNotes; k++)
                {
                    Label note = new Label();
                                                    ControlStyle.LabelStyle(note,   k,
improvedHS.bestHarmoniesNotes[improvedHS.NI - 1, k - 1]);
                    numericsTab.Controls.Add(note);
                }
                saveResults(improvedHS.bestHarmoniesAesthetics[improvedHS.NI - 1]);
                //if (ShowAll == true)
                //    System.Diagnostics.Process.Start("Results.txt");
            }
            if (globalHS != null)
            {
                if (globalHS.ShowAll == true)
                    resultsRichTextBox.Text = globalHS.Results.ToString();
                totalNotes = globalHS.TotalNotes;
                BestHarmonyChart.Series.Add("Aesthetics");
                BestHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
                BestHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                NewHarmonyChart.Series.Add("Aesthetics");
                NewHarmonyChart.Series[0].ChartType = SeriesChartType.Pie;
                NewHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                WorstHarmonyChart.Series.Add("Aesthetics");
                WorstHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
                WorstHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                List<Chart> allDecisionVariables = new List<Chart>();
```

```
                for (int i = 1; i <= globalHS.TotalNotes; i++)
                {
                    Chart noteChart = new Chart();
                    ControlStyle.ChartStyle(noteChart, i);
                    allDecisionVariables.Add(noteChart);
                    Label noteLabel = new Label();
                    ControlStyle.GraphsLabelStyle(noteLabel, i);
                    notesTab.Controls.Add(noteLabel);
                }
                for (int j = 0; j < globalHS.NI; j++)
                {
                                            BestHarmonyChart.Series[0].Points.AddXY(j,
globalHS.bestHarmoniesAesthetics[j]);
                                             NewHarmonyChart.Series[0].Points.AddXY(j,
globalHS.newHarmoniesAesthetics[j]);
                                           WorstHarmonyChart.Series[0].Points.AddXY(j,
globalHS.worstHarmoniesAesthetics[j]);
                    for (int i = 1; i <= globalHS.TotalNotes; i++)
                    {
                                allDecisionVariables[i - 1].Series[0].Points.AddXY(j,
globalHS.bestHarmoniesNotes[j, i - 1]);
                    }
                }
                for (int i = 0; i < globalHS.TotalNotes; i++)
                {
                    notesTab.Controls.Add(allDecisionVariables[i]);
                }
                Label aestheticsLabel = new Label();
                                              ControlStyle.LabelStyle(aestheticsLabel,
globalHS.bestHarmoniesAesthetics[globalHS.NI - 1]);
                numericsTab.Controls.Add(aestheticsLabel);
                for (int k = 1; k <= globalHS.TotalNotes; k++)
                {
                    Label note = new Label();
                                                    ControlStyle.LabelStyle(note,   k,
globalHS.bestHarmoniesNotes[globalHS.NI - 1, k - 1]);
                    numericsTab.Controls.Add(note);
                }
                saveResults(globalHS.bestHarmoniesAesthetics[globalHS.NI - 1]);
                //if (ShowAll == true)
                //    System.Diagnostics.Process.Start("Results.txt");
            }
            if (adaptiveHS != null)
            {
                if (adaptiveHS.ShowAll == true)
                    resultsRichTextBox.Text = adaptiveHS.Results.ToString();
                totalNotes = adaptiveHS.TotalNotes;
                BestHarmonyChart.Series.Add("Aesthetics");
                BestHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
                BestHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                NewHarmonyChart.Series.Add("Aesthetics");
                NewHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
                NewHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                WorstHarmonyChart.Series.Add("Aesthetics");
                WorstHarmonyChart.Series[0].ChartType = SeriesChartType.Line;
                WorstHarmonyChart.Series[0].Color = System.Drawing.Color.Blue;
                List<Chart> allDecisionVariables = new List<Chart>();
                for (int i = 1; i <= adaptiveHS.TotalNotes; i++)
                {
                    Chart noteChart = new Chart();
                    ControlStyle.ChartStyle(noteChart, i);
                    allDecisionVariables.Add(noteChart);
                    Label noteLabel = new Label();
                    ControlStyle.GraphsLabelStyle(noteLabel, i);
                    notesTab.Controls.Add(noteLabel);
```

```
                }
                for (int j = 0; j < adaptiveHS.NI; j++)
                {
                                                BestHarmonyChart.Series[0].Points.AddXY(j,
adaptiveHS.bestHarmoniesAesthetics[j]);
                                                 NewHarmonyChart.Series[0].Points.AddXY(j,
adaptiveHS.newHarmoniesAesthetics[j]);
                                               WorstHarmonyChart.Series[0].Points.AddXY(j,
adaptiveHS.worstHarmoniesAesthetics[j]);
                    for (int i = 1; i <= adaptiveHS.TotalNotes; i++)
                    {
                                allDecisionVariables[i - 1].Series[0].Points.AddXY(j,
adaptiveHS.bestHarmoniesNotes[j, i - 1]);
                    }
                }
                for (int i = 0; i < adaptiveHS.TotalNotes; i++)
                {
                    notesTab.Controls.Add(allDecisionVariables[i]);
                }
                Label aestheticsLabel = new Label();
                                                ControlStyle.LabelStyle(aestheticsLabel,
adaptiveHS.bestHarmoniesAesthetics[adaptiveHS.NI - 1]);
                numericsTab.Controls.Add(aestheticsLabel);
                for (int k = 1; k <= adaptiveHS.TotalNotes; k++)
                {
                    Label note = new Label();
                                                        ControlStyle.LabelStyle(note,   k,
adaptiveHS.bestHarmoniesNotes[adaptiveHS.NI - 1, k - 1]);
                    numericsTab.Controls.Add(note);
                }
                saveResults(adaptiveHS.bestHarmoniesAesthetics[adaptiveHS.NI - 1]);
                //if (ShowAll == true)
                //    System.Diagnostics.Process.Start("Results.txt");
            }
                        BestHarmonyChart.ChartAreas[0].AxisX.MajorGrid.LineColor  =
System.Drawing.Color.LightGray;
                     BestHarmonyChart.ChartAreas[0].AxisX.MajorGrid.LineDashStyle  =
ChartDashStyle.Dot;
                        BestHarmonyChart.ChartAreas[0].AxisY.MajorGrid.LineColor  =
System.Drawing.Color.LightGray;
                     BestHarmonyChart.ChartAreas[0].AxisY.MajorGrid.LineDashStyle  =
ChartDashStyle.Dot;
        }

        private void BackButton_Click(object sender, EventArgs e)
        {
            isBackButtonPressed = true;
            this.Close();
            Configuration.initializeComponents();
            Configuration.Show();
        }

        private void performHarmonySearch()
        {
            if (classicHS != null)
            {
                classicHS.ShowAll = ShowAll;
                classicHS.initializeMemory();
                SearchProgress.Minimum = 0;
                SearchProgress.Maximum = classicHS.NI;
                classicHS.Run(SearchProgress);
            }
            if (improvedHS != null)
            {
                improvedHS.ShowAll = ShowAll;
```

```
                improvedHS.initializeMemory();
                SearchProgress.Minimum = 0;
                SearchProgress.Maximum = improvedHS.NI;
                improvedHS.Run(SearchProgress);
            }
            if (globalHS != null)
            {
                globalHS.ShowAll = ShowAll;
                globalHS.initializeMemory();
                SearchProgress.Minimum = 0;
                SearchProgress.Maximum = globalHS.NI;
                globalHS.Run(SearchProgress);
            }
            if (adaptiveHS != null)
            {
                adaptiveHS.ShowAll = ShowAll;
                adaptiveHS.initializeMemory();
                SearchProgress.Minimum = 0;
                SearchProgress.Maximum = adaptiveHS.NI;
                adaptiveHS.Run(SearchProgress);
            }
        }

        private void GraphsForm_FormClosed(object sender, FormClosedEventArgs e)
        {
            if (isBackButtonPressed)
                Dispose();
            else
                Application.Exit();
        }
    }
```

Harmony.cs

```
public class Harmony

{

   public double[] notes { get; set; }

}
```

ImprovedSearch.cs

```
public class ImprovedSearch : Search
    {
        public float HMCR { get; set; }


        public float PARmin { get; set; }
        public float PARmax { get; set; }
        public double BWmin { get; set; }
        public double BWmax { get; set; }
```

```
        private static ImprovedSearch instance = null;

        private ImprovedSearch()
        {
            initializeMemory();
        }

        public static ImprovedSearch Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new ImprovedSearch();
                }
                return instance;
            }
        }

        public void initializeMemory()
        {
            Results = new StringBuilder();
            base.bestHarmoniesAesthetics = new double[NI];
            base.newHarmoniesAesthetics = new double[NI];
            base.worstHarmoniesAesthetics = new double[NI];
            base.bestHarmoniesNotes = new double[NI, TotalNotes];
            Memory = null;
            Memory = new Harmony[HMSize];
            for (int i = 0; i < HMSize; i++)
                Memory[i] = getRandomHarmony();

            sortMemory();
        }

        public void Run(ProgressBar progressBar)
        {
                    for (int currentImprovisation = 0; currentImprovisation < NI;
currentImprovisation++)
            {
                Harmony newHarmony = new Harmony();
                newHarmony.notes = new double[TotalNotes];
                for (int currentNote = 0; currentNote < TotalNotes; currentNote++)
                {
                    float randomFloat = Randomizer.getRandomFloat(0.0f, 1.0f);
                    if (randomFloat <= HMCR)
                    {
                                                                int randomHarmony =
Convert.ToInt32(Randomizer.getRandomDouble(0, HMSize - 1));
                                                        newHarmony.notes[currentNote] =
Memory[randomHarmony].notes[currentNote];
                        adjustPitch(newHarmony, currentNote, currentImprovisation);
                    }
                    else
                    {
                                                        newHarmony.notes[currentNote] =
Randomizer.getRandomDouble(MinimumValues[currentNote], MaximumValues[currentNote]);
                    }
                }
                updateMemory(newHarmony, currentImprovisation);
                //if (currentImprovisation == 249)
                //{
```

```
                //    countDuplicates();
                //}
                                base.bestHarmoniesAesthetics[currentImprovisation]  =
getHarmonyAesthetics(Memory[0]);
                                 base.newHarmoniesAesthetics[currentImprovisation]  =
getHarmonyAesthetics(newHarmony);
                                base.worstHarmoniesAesthetics[currentImprovisation]  =
getHarmonyAesthetics(Memory[HMSize - 1]);
                for (int i = 0; i < TotalNotes; i++)
                {
                                 base.bestHarmoniesNotes[currentImprovisation,  i]  =
Memory[0].notes.ElementAt(i);
                }
                if (ShowAll == true)
                {
                    base.writeResults(currentImprovisation);
                }

                progressBar.Value = currentImprovisation;
                                int  percent  =  (int)(((double)(progressBar.Value  -
progressBar.Minimum) / (double)(progressBar.Maximum - progressBar.Minimum)) * 100);
                string progressMessage = "Please wait... Harmony Search in progress.
" + percent.ToString() + "%" + " completed.";
                    ControlStyle.SetProgressBarText(progressBar,  progressMessage,
ControlStyle.ProgressBarTextLocation.Centered, Color.Black, new Font("Arial", 16));
            }
        }

                 private  void  adjustPitch(Harmony  newHarmony,  int  index,  int
currentImprovisation)
        {
            float randomFloat = Randomizer.getRandomFloat(0.0f, 1.0f);
            if (randomFloat <= getPAR(currentImprovisation))
            {
                randomFloat = Randomizer.getRandomFloat(-10.0f, 10.0f);
                if (randomFloat < 0)
                {
                            newHarmony.notes[index]  +=  Randomizer.getRandomDouble(0,
getBandwidth(currentImprovisation));
                    newHarmony.notes[index] = restrictNote(newHarmony.notes[index],
index);
                }
                else if (randomFloat >= 0)
                {
                            newHarmony.notes[index]  -=  Randomizer.getRandomDouble(0,
getBandwidth(currentImprovisation));
                    newHarmony.notes[index] = restrictNote(newHarmony.notes[index],
index);
                }
            }
        }

        private float getPAR(int currentImprovisation)
        {
            float newPAR = PARmin + ((PARmax - PARmin) / NI) * currentImprovisation;
            return newPAR;
        }

        private double getBandwidth(int currentImprovisation)
        {
            double newBW = BWmax - ((BWmax - BWmin) / NI) * currentImprovisation;
            return newBW;
        }
```

```
    }
```

Parameters.cs

```
public class Parameters
    {
        public string Objective { get; set; }
        public bool ShowAll { get; set; }
        public OptimizationGoal Optimum { get; set; }
        public double[] MaxValues { get; set; }
        public double[] MinValues { get; set; }
        public HarmonySearchVariant Variant { get; set; }
        public int NI { get; set; }
        public int HMS { get; set; }
        public double HMCR { get; set; }
        public double PAR { get; set; }
        public double BW { get; set; }
        public double PARmin { get; set; }
        public double PARmax { get; set; }
        public double BWmin { get; set; }
        public double BWmax { get; set; }
    }
```

Randomizer.cs

```
public static class Randomizer
    {
        private static readonly Random random = new Random();
        private static readonly object syncLock = new object();

        public static int getRandomInteger(int min, int max)
        {
            lock (syncLock)
            {
                return random.Next(min, max);
            }
        }

        public static double getRandomDouble(double min, double max)
        {
            lock (syncLock)
            {
                return random.NextDouble() * (max - min) + min;
            }
        }

        public static float getRandomFloat(float min, float max)
        {
            lock (syncLock)
            {
                return (float)random.NextDouble() * (max - min) + min;
            }
        }
    }
```

Search.cs

```
public enum HarmonySearchVariant
    {
        Classic, Improved, GlobalBest, SelfAdaptive
    }
    public enum OptimizationGoal
    {
        Max, Min, MinAbs
    }
    public abstract class Search
    {
        public int NI { get; set; } //Number of Improvisations
        public double[] MinimumValues { get; set; }
        public double[] MaximumValues { get; set; }
        public int TotalNotes { get; set; }
        public int HMSize { get; set; }

        public double[] bestHarmoniesAesthetics { get; set; }
        public double[] newHarmoniesAesthetics { get; set; }
        public double[] worstHarmoniesAesthetics { get; set; }
        public double[,] bestHarmoniesNotes { get; set; }
        public StringBuilder Results { get; set; }
        public bool ShowAll { get; set; }

        protected Harmony[] Memory;

        public Expression Objective { get; set; }
        public OptimizationGoal Optimum { get; set; }
```

```
        protected virtual Harmony getRandomHarmony()
        {
            Harmony hrm = new Harmony();
            hrm.notes = new double[TotalNotes];
            for (int i = 0; i < TotalNotes; i++)
                        hrm.notes[i] = Randomizer.getRandomDouble(MinimumValues[i],
MaximumValues[i]);

            return hrm;
        }

        protected double getHarmonyAesthetics(Harmony harmony)
        {
            int j = 0;
            for(int i = 0; i < TotalNotes; i++)
            {
                j = i + 1;
                Objective.Parameters["x" + j] = harmony.notes[i];
            }
            return (double)Objective.Evaluate();
        }

        protected void sortMemory()
        {
            Harmony tempHar = new Harmony();
            for (int i = 0; i < Memory.Length; i++)
            {
                for (int j = i + 1; j < HMSize; j++)
                {
                    if (Optimum.Equals(OptimizationGoal.Max))
                    {
                                                if (getHarmonyAesthetics(Memory[i]) <
getHarmonyAesthetics(Memory[j]))
                        {
```

```
                            tempHar = Memory[i];
                            Memory[i] = Memory[j];
                            Memory[j] = tempHar;
                        }
                    }
                    if (Optimum.Equals(OptimizationGoal.Min))
                    {
                                                if (getHarmonyAesthetics(Memory[i]) >
getHarmonyAesthetics(Memory[j]))
                        {
                            tempHar = Memory[i];
                            Memory[i] = Memory[j];
                            Memory[j] = tempHar;
                        }
                    }
                    if (Optimum.Equals(OptimizationGoal.MinAbs))
                    {
                                            if (Math.Abs(getHarmonyAesthetics(Memory[i])) >
Math.Abs(getHarmonyAesthetics(Memory[j])))
                        {
                            tempHar = Memory[i];
                            Memory[i] = Memory[j];
                            Memory[j] = tempHar;
                        }
                    }
                }
            }
        }

        protected void updateMemory(Harmony newHar, int currentIteration)
        {
            if (Optimum.Equals(OptimizationGoal.Max))
            {
                                                if (getHarmonyAesthetics(newHar) >
getHarmonyAesthetics(Memory[HMSize - 1]))
```

```
                {
                    Memory[HMSize - 1] = newHar;
                    sortMemory();
                }
            }
            if (Optimum.Equals(OptimizationGoal.Min))
            {
                                                    if   (getHarmonyAesthetics(newHar)   <
getHarmonyAesthetics(Memory[HMSize - 1]))
                {
                    Memory[HMSize - 1] = newHar;
                    sortMemory();
                }
            }
            if (Optimum.Equals(OptimizationGoal.MinAbs))
            {
                                          if   (Math.Abs(getHarmonyAesthetics(newHar))   <
Math.Abs(getHarmonyAesthetics(Memory[HMSize - 1])))
                {
                    Memory[HMSize - 1] = newHar;
                    sortMemory();
                }
            }
        }


        protected double restrictNote(double note, int index)
        {
            if (note > MaximumValues[index])
                return MaximumValues[index];
            else if (note < MinimumValues[index])
                return MinimumValues[index];
            else
                return note;
        }
```

```
        public void writeResults(int currentImprovisation)
        {
            Results.Append("-------------------------------------------------------------
-------------------------------------");
                  Results.Append(Environment.NewLine + "Improvisation Number: " +
currentImprovisation + Environment.NewLine);
            Results.Append("-------------------------------------------------------------
-------------------------------------");
            for (int i = 0; i < HMSize; i++)
            {
                Results.Append(Environment.NewLine);
                Results.Append(i + " Harmony: ");
                //Results += Environment.NewLine;
                for (int j = 0; j < TotalNotes; j++)
                {
                    Results.Append("\t");
                                        Results.Append("Note " + j + ": " +
Math.Round(Memory[i].notes[j], 3));
                }
                //Results += Environment.NewLine;
                                          Results.Append("\t Aesthetics: " +
Math.Round(getHarmonyAesthetics(Memory[i]), 3));
                //Results += Environment.NewLine;
            }
            Results.Append(Environment.NewLine);
            if (currentImprovisation == (NI - 1))
            {
                saveResultsToFile();
            }
        }

        private void saveResultsToFile()
        {
            string path = @"Results.txt";
```

```
            File.WriteAllText(path, Results.ToString());
        }


        protected int countDuplicates()
        {
            int duplicatesCounter = 0;
            List<Harmony> selectedHarmonies = new List<Harmony>();
            bool harmonyExists;


            for (int i = 0; i < HMSize - 1; i++)
            {
                harmonyExists = false;
                for(int k = 0; k < selectedHarmonies.Count; k++)
                {
                                                    if(getHarmonyAesthetics(Memory[i])   ==
getHarmonyAesthetics(selectedHarmonies[k]))
                    {
                        harmonyExists = true;
                        break;
                    }
                }
                if (harmonyExists == true)
                    continue;
                selectedHarmonies.Add(Memory[i]);
                for (int j = i+1; j < HMSize; j++)
                {
                                                    if(getHarmonyAesthetics(Memory[i])   ==
getHarmonyAesthetics(Memory[j]))
                    {
                        duplicatesCounter++;
                    }
                }
            }
            return duplicatesCounter;
```

```
        }


    }
```

SelfAdaptive.cs

```
public class SelfAdaptiveSearch : Search
    {
        private List<double> minVariables;
        private List<double> maxVariables;
        public float HMCR { get; set; }
        public float PAR { get; set; }
        public double BW { get; set; }

        private static SelfAdaptiveSearch instance = null;

        private SelfAdaptiveSearch()
        {
            initializeMemory();
        }

        public static SelfAdaptiveSearch Instance
        {
            get
            {
                if (instance == null)
                {
                    instance = new SelfAdaptiveSearch();
                }
                return instance;
            }
        }

        //List<double> harmony = new List<double>(TotalVars);
        //int randomIntNumber = 0;

        protected override Harmony getRandomHarmony()
        {
            Harmony hrm = new Harmony();
            hrm.notes = new double[TotalNotes];
            for (int i = 0; i < TotalNotes; i++)
                        hrm.notes[i] = Randomizer.getRandomDouble(MinimumValues[i],
MaximumValues[i]);

            checkMaxVariables(hrm);
            checkMinVariables(hrm);

            return hrm;
        }
```

```
        public void initializeMemory()
        {
            Results = new StringBuilder();
            base.bestHarmoniesAesthetics = new double[NI];
            base.newHarmoniesAesthetics = new double[NI];
            base.worstHarmoniesAesthetics = new double[NI];
            base.bestHarmoniesNotes = new double[NI, TotalNotes];
            Memory = new Harmony[HMSize];
            //TODO:
            initializeExtremeVariables();
            for (int i = 0; i < HMSize; i++)
                Memory[i] = getRandomHarmony();


            sortMemory();
        }


        public void Run(ProgressBar progressBar)
        {
                    for (int currentImprovisation = 0; currentImprovisation < NI;
currentImprovisation++)
            {
                Harmony newHarmony = new Harmony();
                newHarmony.notes = new double[TotalNotes];
                for (int currentNote = 0; currentNote < TotalNotes; currentNote++)
                {
                    float randomFloat = Randomizer.getRandomFloat(0.0f, 1.0f);
                    if (randomFloat <= HMCR)
                    {
                                                                int randomHarmony =
Convert.ToInt32(Randomizer.getRandomDouble(0, HMSize - 1));
                                                        newHarmony.notes[currentNote] =
Memory[randomHarmony].notes[currentNote];
                        updatePAR(currentImprovisation);
                        adjustPitch(newHarmony, currentNote);
                    }
                    else
                    {
                                                        newHarmony.notes[currentNote] =
Randomizer.getRandomDouble(MinimumValues[currentNote], MaximumValues[currentNote]);
                    }
                }
                updateMemory(newHarmony, currentImprovisation);
                                    base.bestHarmoniesAesthetics[currentImprovisation] =
getHarmonyAesthetics(Memory[0]);
                                     base.newHarmoniesAesthetics[currentImprovisation] =
getHarmonyAesthetics(newHarmony);
                                    base.worstHarmoniesAesthetics[currentImprovisation] =
getHarmonyAesthetics(Memory[HMSize - 1]);
                for (int i = 0; i < TotalNotes; i++)
                {
                                     base.bestHarmoniesNotes[currentImprovisation, i] =
Memory[0].notes.ElementAt(i);
                }


                if (ShowAll == true)
                {
                    base.writeResults(currentImprovisation);
                }


                progressBar.Value = currentImprovisation;
                                    int percent = (int)(((double)(progressBar.Value -
progressBar.Minimum) / (double)(progressBar.Maximum - progressBar.Minimum)) * 100);
```

```
                string progressMessage = "Please wait... Harmony Search in progress.
" + percent.ToString() + "%" + " completed.";
                    ControlStyle.SetProgressBarText(progressBar, progressMessage,
ControlStyle.ProgressBarTextLocation.Centered, Color.Black, new Font("Arial", 16));
            }
        }

        private void adjustPitch(Harmony newHarmony, int index)
        {
            //TODO: Must adjust based on the best and worst decision-variables
            float randomFloat = Randomizer.getRandomFloat(0.0f, 1.0f);
            if (randomFloat <= PAR)
            {
                randomFloat = Randomizer.getRandomFloat(-10.0f, 10.0f);
                if (randomFloat < 0)
                                newHarmony.notes[index] += (maxVariables[index] -
newHarmony.notes[index]) * Randomizer.getRandomDouble(0.0, 1.0);
                else if (randomFloat >= 0)
                            newHarmony.notes[index] -= (newHarmony.notes[index] -
minVariables[index] ) * Randomizer.getRandomDouble(0.0, 1.0);

                newHarmony.notes[index] = restrictNote(newHarmony.notes[index],
index);
            }
        }

        private void updatePAR(int currentImprovisation)
        {
            float newPAR = 1 - ((1 / NI) * currentImprovisation);
            PAR = newPAR;
        }

        private void checkMaxVariables(Harmony newHarmony)
        {
            for(int i = 0; i < TotalNotes; i++)
            {
                if (newHarmony.notes[i] > maxVariables[i])
                    maxVariables[i] = newHarmony.notes[i];
            }
        }

        private void checkMinVariables(Harmony newHarmony)
        {
            for (int i = 0; i < TotalNotes; i++)
            {
                if (newHarmony.notes[i] < minVariables[i])
                    minVariables[i] = newHarmony.notes[i];
            }
        }

        private void initializeExtremeVariables()
        {
            minVariables = new List<double>();
            maxVariables = new List<double>();
            for(int i=0; i<TotalNotes; i++)
            {
                minVariables.Add(MaximumValues[i]);
                maxVariables.Add(MinimumValues[i]);
            }
        }
    }
```