



Σχολή Θετικών Επιστημών και Τεχνολογίας

Πρόγραμμα Σπουδών Πληροφορικής

Πτυχιακή Εργασία

Υλοποίηση Grey Wolf Optimizer (GWO) Algorithm

Χρήστος Α. Σμαρλαμάκης

Επιβλέπων καθηγητής: Δρ. Ευστράτιος Φ. Γεωργόπουλος

Πάτρα, Ιούλιος 2025

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή Χρήστου Α. Σμαρλαμάκη που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.

Υλοποίηση Grey Wolf Optimizer (GWO) Algorithm

Χρήστος Α. Σμαρλαμάκης

Επιτροπή Επίβλεψης Πτυχιακής Εργασίας

Επιβλέπων Καθηγητής:

Δρ. Ευστράτιος Φ. Γεωργόπουλος

Καθηγητής

Τμήμα Διοίκησης

Επιχειρήσεων & Οργανισμών

Πανεπιστήμιο Πελοποννήσου

Συνεργαζόμενο Εκπαιδευτικό Προσωπικό
(ΣΕΠ)

Ελληνικό Ανοικτό Πανεπιστήμιο (ΕΑΠ)

Συν-Επιβλέπων Καθηγητής:

Δρ. Γρηγόριος Ν. Μπεληγιάννης

Καθηγητής

Τμήμα Επιστήμης & Τεχνολογίας
Τροφίμων

Πανεπιστήμιο Πατρών

Συν-Επιβλέπουσα Καθηγήτρια:

Δρ. Αντωνία Στεφανή

Επίκουρη Καθηγήτρια

Τμήμα Διοικητικής
Επιστήμης & Τεχνολογίας
Πανεπιστήμιο Πατρών

Πάτρα, Ιούλιος 2025

*Αφιερώνω την παρούσα εργασία στην οικογένειά μου, που αποτελεί αδιάλειπτη πηγή ηθικής
υποστήριξης και ενθάρρυνσης, θεμέλιο στην συνεχή αναζήτηση της γνώσης και της
αλήθειας.*

*Ἐν οἶδα ὅτι οὐδὲν οἶδα.
Σωκράτης.*

Περίληψη

Η παρούσα πτυχιακή εργασία έχει ως αντικείμενο την παρουσίαση του μεταερευτικού αλγορίθμου με τίτλο Βελτιστοποιητής Γκρίζων Λύκων (Grey Wolf Optimizer - GWO), ο οποίος ανήκει στη γενικότερη κατηγορία των Μεταερευτικών Αλγορίθμων Νοημοσύνης Σμήνους (Swarm Intelligence Metaheuristic Algorithms). Οι συγκεκριμένοι αλγόριθμοι βρίσκουν εφαρμογή σε πληθώρα πεδίων, όπως η βελτιστοποίηση συνδυαστικών προβλημάτων, η εκπαίδευση τεχνητών νευρωνικών δικτύων, η ρομποτική, καθώς και σε χρηματοοικονομικές εφαρμογές, όπως η πρόβλεψη της απόδοσης χρηματοοικονομικών μέσων.

Ο αλγόριθμος Grey Wolf Optimizer προτάθηκε αρχικά το 2013, ενώ το 2019 παρουσιάστηκε μια βελτιωμένη έκδοσή του. Ο αλγόριθμος είναι εμπνευσμένος από τη συμπεριφορά και τον τρόπο διαβίωσης των γκρίζων λύκων, ιδιαίτερα από τις στρατηγικές κυνηγιού και τη δυναμική της αγέλης τους, οι οποίες χρησιμοποιούνται ως βασικοί μηχανισμοί για την εύρεση τροφής, καθώς και από την εξερεύνηση και την αναπαραγωγική συμπεριφορά τους.

Η υλοποίηση του αλγορίθμου πραγματοποιήθηκε στη γλώσσα προγραμματισμού Python με τη χρήση διαφόρων κατάλληλων βιβλιοθηκών και ο κώδικας παρατίθεται στο Παράρτημα «Α» της παρούσας εργασίας. Στο πλαίσιο της εργασίας πραγματοποιείται συγκριτική αξιολόγηση της υλοποίησης, τόσο με άλλους αλγορίθμους όσο και μέσω παραμετρικής ανάλυσης, με σκοπό την εξέταση της αποδοτικότητας της σε διαφορετικά προβλήματα βελτιστοποίησης.

Λέξεις – Κλειδιά

Μεταερευτικοί Αλγόριθμοι, Υπολογιστική Νοημοσύνη, Νοημοσύνη Σμήνους, Αλγόριθμος Grey Wolf Optimizer, Python.

Abstract

This thesis presents the Grey Wolf Optimizer (GWO), a metaheuristic algorithm that belongs to the broader family of Swarm Intelligence algorithms. These algorithms have found applications in various fields, including combinatorial optimization, artificial neural network training, robotics, and financial forecasting, such as predicting the performance of financial assets.

The Grey Wolf Optimizer was first introduced in 2013, and an improved version was proposed in 2019. It is inspired by the social behavior and hunting strategies of grey wolves, particularly their hierarchical structure of their packs and their cooperative techniques for hunting, exploration, and reproduction.

The algorithm has been implemented in the Python programming language using various scientific libraries. The complete source code is included in Appendix «A». Furthermore, this thesis provides a comprehensive performance evaluation of the implementation. The algorithm is benchmarked against other well-known metaheuristics and a sensitivity analysis of its key parameters is conducted, in order to assess its effectiveness and robustness across various optimization problems.

Keywords

Metaheuristic Algorithms, Computational Intelligence, Swarm Intelligence, Grey Wolf Optimizer Algorithm, Python.

Περιεχόμενα

Περίληψη.....	v
Abstract	vii
Περιεχόμενα	viii
Κατάλογος Εικόνων	xi
Κατάλογος Πινάκων	xiii
Κατάλογος Γραφημάτων.....	xiv
Συντομογραφίες & Ακρωνύμια.....	xv
1. Εισαγωγή.....	1
1.1 Στόχοι Υλοποίησης	1
1.2 Διάρθρωση Εργασίας	1
2. Θεωρητικό Υπόβαθρο	4
2.1 Εισαγωγή στην Βελτιστοποίηση	4
2.2 Μεταευρετικοί Αλγόριθμοι	6
2.2.1 Ορισμός και Κατηγορίες	7
2.2.2 Χαρακτηριστικά Μεταευρετικών	8
2.2.3 Χρονική Πολυπλοκότητα	9
2.3 Εξελικτικοί Αλγόριθμοι	9
2.3.1 Ορισμός	10
2.3.2 Κύρια Χαρακτηριστικά	10
2.4 Νοημοσύνη Σμήνους	11
2.4.1 Ορισμός	11
2.4.2 Εφαρμογές και Χαρακτηριστικά	11
2.5 Αλγόριθμοι Βελτιστοποίησης Βασισμένοι στους Λύκους.....	13
2.5.1 Αλγόριθμος Αναζήτησης των Λύκων - WSA	13
2.5.2 Αλγόριθμοι βασισμένοι στον WSA	18
3. Ο Αλγόριθμος Grey Wolf Optimizer	23
3.1 Εισαγωγή.....	23
3.2 Χαρακτηριστικά των Γκρίζων Λύκων	23
3.3 Μαθηματικό Μοντέλο του Αλγορίθμου	26
3.3.1 Μοντελοποίηση της Περικύκλωσης του Θηράματος	27
3.3.2 Μοντελοποίηση της Αναζήτησης για Θήραμα (Εξερεύνηση).....	29

3.3.3 Μοντελοποίηση της Θηρευτικής Διαδικασίας (Κυνήγι)	30
3.3.4 Μοντελοποίηση της Επίθεσης στο Θήραμα (Εκμετάλλευση)	32
3.3.5 Σημαντικά Σημεία και Ψευδοκώδικας του Αλγορίθμου	33
3.4 Περιορισμοί και Μειονεκτήματα του Αλγορίθμου	38
3.5 Παραλλαγές και Εφαρμογές του Αλγορίθμου	40
3.5.1 Τροποποιημένες Εκδόσεις του GWO	40
3.5.2 Υβριδικές Εκδόσεις του GWO	42
3.5.3 Στρατηγική Παραλληλισμού (Parallelism)	44
3.5.4 Multi-objective GWO (MOGWO)	45
3.5.5 Εφαρμογές των Παραλλαγών του GWO	45
4. Σχεδιασμός – Υλοποίηση του Αλγορίθμου	48
4.1 Σχετικά με τη Γλώσσα Υλοποίησης	48
4.2 Βιβλιοθήκες και πακέτα	49
4.2.1 Η Βιβλιοθήκη NumPy	49
4.2.2 Η Βιβλιοθήκη SciPy και το Πακέτο SciPy.stats	49
4.2.3 Η Βιβλιοθήκη Matplotlib	50
4.2.4 Η Βιβλιοθήκη Time	51
4.2.5 Το Πακέτο Concurrent.futures	52
4.3 Λογισμικά και Εργαλεία	52
4.3.1 Το Οικοσύστημα Anaconda	52
4.3.2 Το IDE Spyder	54
4.3.3 Το Google Colaboratory (Colab)	54
4.4 Παρουσίαση του Αλγορίθμου	55
5. Αξιολόγηση του Αλγορίθμου	64
5.1 Εισαγωγή στις Συναρτήσεις Αξιολόγησης	64
5.2 Συναρτήσεις Αξιολόγησης για τον GWO	66
5.2.1 Συνάρτηση Ackley (Ackley Function)	66
5.2.2 Συνάρτηση Griewank (Griewank Function)	68
5.2.3 Συνάρτηση Rastrigin (Rastrigin Function)	69
5.2.4 Συνάρτηση Rosenbrock (Rosenbrock Function)	69
5.2.5 Συνάρτηση Sphere (Sphere Function)	70
5.2.6 Συνάρτηση De Jong No5 ή Shekel's Foxholes (De Jong Function / Shekel's Foxholes Function)	71

5.2.7 Συνάρτηση Goldstein-Price (Goldstein-Price Function).....	72
6. Πειραματική Εφαρμογή και Αποτελέσματα GWO.....	73
6.1 Εφαρμογή Αλγορίθμου GWO σε Συναρτήσεις Δοκιμής.....	73
6.1.1 Σύγκριση Αποτελεσμάτων Με Άλλους Αλγορίθμους	73
6.1.2 Μελέτη και Ρύθμιση Παραμέτρων.....	75
6.1.2.1 Πρώτο Πείραμα.....	76
6.1.2.2 Δεύτερο Πείραμα	77
6.1.2.3 Τρίτο Πείραμα.....	78
6.1.2.4 Τέταρτο Πείραμα	79
6.1.2.5 Πέμπτο Πείραμα	82
6.1.2.6 Έκτο Πείραμα	86
6.1.2.7 Έβδομο Πείραμα	90
6.1.2.8 Όγδοο Πείραμα	92
6.2 Συμπεράσματα Αξιολόγησης και Παραμετροποίησης GWO.....	95
7. Βιβλιογραφία.....	97
Παράρτημα Α «Κώδικας Αλγορίθμου»	103
A-1 Αρχείο gui.py	103
A-2 Αρχείο main.py	139
A-3 Αρχείο gwo.py	141
A-4 Αρχείο benchmark_function.py	149
A-5 Αρχείο plot_graph.py	153
Παράρτημα Β «Χαρακτηριστικά Υπολογιστικού Συστήματος Δοκιμών»	155
Παράρτημα Γ «Γραφικό Περιβάλλον Διεπαφής (GUI) »	157
Γ-1 Εισαγωγή	157
Γ-2 Κύρια Οθόνη και Λειτουργίες.....	157
Γ-3 Οδηγίες Χρήσης	159

Κατάλογος Εικόνων

Εικόνα 1: Εξελικτικός Κύκλος.....	10
Εικόνα 2: Σμήνος φλαμίνγκο σε σχηματισμό. (Author: Nikunj Vasoya – own work, CC BY-SA 3.0. https://creativecommons.org/licenses/by-sa/3.0/deed.en .).....	12
Εικόνα 3: Στιγμιότυπο Αλγορίθμου Αναζήτησης Λύκων (WSA).	20
Εικόνα 4: Αγέλη Γκρίζων Λύκων (Author: ALEXANDRE - stock.adobe.com. https://wildark.org/wp-content/uploads/2019/05/AdobeStock_200361935.jpg .).....	23
Εικόνα 5: Ιεραρχία Αγέλης Γκρίζων Λύκων.	25
Εικόνα 6: Θηρευτική Διαδικασία Γκρίζων Λύκων [42]	25
Εικόνα 7: Τυπικές κυνηγετικές συμπεριφορές της αγέλης των λύκων. Ένας βίσωνας στο Εθνικό Πάρκο Yellowstone καταδιώκεται από μια αγέλη εννέα λύκων. (Α) Προσέγγιση, παρακολούθηση και προσέγγιση. (Β-Δ) Καταδίωξη, παρενόχληση και ελιγμός περικύκλωσης. (Ε) [41].....	26
Εικόνα 8: Δισδιάστατα Διανύσματα Θέσεων Λύκου και Θηράματος.	28
Εικόνα 9: Θέσεις λύκων και θηράματος κατά την τελική φάση του κυνηγιού.	32
Εικόνα 10: Επίθεση στο θήραμα (αριστερά) και αναζήτησης για θήραμα (δεξιά). [40]....	33
Εικόνα 11: Διαδικασία ομαδικού κυνηγιού γκρίζων λύκων σε αγέλη (α) εντοπισμός θηράματος, (b) περικύκλωση θηράματος και (c) επίθεση θηράματος. [44].....	36
Εικόνα 12: Διάγραμμα Ροής του GWO. [43].....	37
Εικόνα 13: Διάγραμμα Εφαρμογών του GWO. [47]	47
Εικόνα 14: Διάγραμμα Απεικόνισης Θέσεων Πρακτόρων Αναζήτησης στο Επίπεδο.	51
Εικόνα 15: Στιγμιότυπο Εκτέλεσης Κώδικα Στο IDE Spyder.	54
Εικόνα 16: Στιγμιότυπο Εκτέλεσης Κώδικα Στο Google Colab.....	55
Εικόνα 17: Διάγραμμα Ροής GWO.....	58
Εικόνα 18: Γράφημα Συνάρτησης Ackley.	67
Εικόνα 19: Γράφημα Κάτοψης Συνάρτησης Ackley	67
Εικόνα 20: Γράφημα Συνάρτησης Griewank.....	68
Εικόνα 21: Γραφήματα Σκιάς Συνάρτησης Griewank.....	68
Εικόνα 22: Γραφήματα Συνάρτησης Rastrigin	69
Εικόνα 23: Γραφήματα Συνάρτησης Rosenbrock.....	70
Εικόνα 24: Γραφήματα Συνάρτησης Sphere.....	71
Εικόνα 25: Γραφήματα Συνάρτησης De Jong No5.....	71

Εικόνα Γ - 1 : Κύρια Οθόνη GWO Simulator.....	157
Εικόνα Γ - 2 : Πλαίσιο Εισαγωγής Παραμέτρων.....	159
Εικόνα Γ - 3 : Παράδειγμα Γραφήματος 3D View της Διαδικασίας Βελτιστοποίησης ...	160
Εικόνα Γ - 4 : Παράδειγμα Γραφήματος Top View.....	160
Εικόνα Γ - 5 : Παράδειγμα Καμπύλης Σύγκλισης	161
Εικόνα Γ - 6 : Επιλογές Θέματος και Φόντου Γραφημάτων	161
Εικόνα Γ - 7 : Απεικόνιση Πληροφοριών στην περιοχή Log	162

Κατάλογος Πινάκων

Πίνακας 1 : Τιμές Παραμέτρων Αλγορίθμου.....	73
Πίνακας 2 : Αποτελέσματα σε Unimodal Συναρτήσεις F1 (Sphere) και F5 (Rosenbrock)	74
Πίνακας 3 : Αποτελέσματα σε Multimodal Συναρτήσεις F9 (Rastrigin), F10 (Ackley) και F11 (Griewank).....	74
Πίνακας 4 : Αποτελέσματα σε Multimodal Δύο Διαστάσεων Συναρτήσεις F14 (Shekel's Foxholes – De Jong N5) και F18 (Goldstine-Price).....	74
Πίνακας 5 : Τιμές Παραμέτρων Πρώτου Πειράματος.....	76
Πίνακας 6 : Αποτελέσματα Πρώτου Πειράματος για τις Συναρτήσεις Sphere και Ackley	76
Πίνακας 7 : Τιμές Παραμέτρων Δεύτερου Πειράματος.....	77
Πίνακας 8 : Αποτελέσματα Δεύτερου Πειράματος.....	77
Πίνακας 9 : Τιμές Παραμέτρων Τρίτου Πειράματος.....	78
Πίνακας 10 : Αποτελέσματα Τρίτου Πειράματος.....	78
Πίνακας 11 : Τιμές Παραμέτρων Τρίτου Πειράματος.....	79
Πίνακας 12 : Αποτελέσματα Τέταρτου Πειράματος.....	79
Πίνακας 13 : Τιμές Παραμέτρων Πέμπτου Πειράματος.....	82
Πίνακας 14 : Αποτελέσματα Πέμπτου Πειράματος.....	83
Πίνακας 15 : Τιμές Παραμέτρων Έκτου Πειράματος.....	86
Πίνακας 16 : Αποτελέσματα Έκτου Πειράματος.....	87
Πίνακας 17 : Τιμές Παραμέτρων Έβδομου Πειράματος	90
Πίνακας 18 : Τιμές Αποτελεσμάτων Έβδομου Πειράματος	91
Πίνακας 19 : Τιμές Παραμέτρων Όγδοου Πειράματος.....	92
Πίνακας 20 : Τιμές Αποτελεσμάτων Όγδοου Πειράματος	92
Πίνακας B - 1 : Τεχνικά Χαρακτηριστικά CPU.....	155
Πίνακας B - 2 : Τεχνικά Χαρακτηριστικά RAM	156
Πίνακας B - 3 : Τεχνικά Χαρακτηριστικά GPU	156

Κατάλογος Γραφημάτων

Γράφημα 1 : Γράφημα Απεικόνισης της Επίδρασης του Πλήθους των Πρακτόρων Αναζήτησης.....	80
Γράφημα 2 : Γράφημα Επίδρασης Μεγέθους Πληθυσμού στον Χρόνο Εκτέλεσης.....	81
Γράφημα 3 : Γράφημα Επίδρασης Κριτηρίου Τερματισμού Τυπικής Απόκλισης στην Μέση Ελάχιστη Τιμή Καταλληλότητας	84
Γράφημα 4 : Γράφημα Επίδρασης Κριτηρίου Τερματισμού Τυπικής Απόκλισης στον χρόνο εκτέλεσης	85
Γράφημα 5 : Επίδραση Επαναλήψεων Αλγορίθμου στην Μέση Ελάχιστη Καταλληλότητα ανα Συνάρτηση.....	88
Γράφημα 6 : Επίδραση Πλήθος Επαναλήψεων στον Μέσο Χρόνο Εκτέλεσης	89
Γράφημα 7 : Επίδραση Μεγέθους Πληθυσμού και Επαναλήψεων στην Μέση Ελάχιστη Τιμή Καταλληλότητας ανα Συνάρτηση Αξιολόγησης.....	93
Γράφημα 8 : Επίδραση Πλήθους Πρακτόρων και Επαναλήψεων στο Μέσο Χρόνο Εκτέλεσης.....	94

Συντομογραφίες & Ακρωνύμια

BM	Brownian Motion – Μπραουνιανή Κίνηση
CPU	Central Processing Unit – Κεντρική Μονάδα Επεξεργασίας
DE	Differential Evolution – Διαφορικός Εξελικτικός Αλγόριθμος
GPU	Graphics Processing Unit – Μονάδα Επεξεργασίας Γραφικών
GSA	Gravitational Search Algorithm – Αλγόριθμος Βαρυτικής Αναζήτησης
GWO	Grey Wolf Optimizer – Βελτιστοποιητής Γκρίζων Λύκων
NFL	No Free Lunch – Θεώρημα Χωρίς Δωρεάν Γεύμα
PSO	Particle Swarm Optimizer – Βελτιστοποιητής Σμήνους Σωματιδίων
RADAR	Radio Detection and Range – Ραδιοεντοπιστής
TPU	Tensor Processing Unit – Μονάδα Επεξεργασίας Τανυστών
TSP	Traveling Salesman Problem – Πρόβλημα Περιοδεύοντος Πωλητή.
VRP	Vehicle Routing Problem – Πρόβλημα Δρομολόγησης Οχημάτων
WSA	Wolf Search Algorithm – Αλγόριθμος Αναζήτησης Λύκων
WPA	Wolf Pack Algorithm – Αλγόριθμος Αγέλης Λύκων

1. Εισαγωγή

1.1 Στόχοι Υλοποίησης

Στόχος της παρούσας εργασίας είναι η αξιολόγηση της απόδοσης της προγραμματιστικής υλοποίησης με χρήση της γλώσσας προγραμματισμού python, του μεταερευτικού αλγορίθμου νοημοσύνης σμήνους Grey Wolf Optimizer. Η απόδοση του αλγορίθμου και η αποτελεσματικότητα στην εξεύρεση της βέλτιστης λύσης θα συγκριθεί με άλλους ευρέως διαδεδομένους αλγορίθμους σε γνωστά προβλήματα δοκιμής. Από την σύγκριση των αποτελεσμάτων θα εξαχθούν συμπεράσματα αναφορικά με την επιτυχία της υλοποίησης μας τόσο σε γενικό όσο και σε ειδικό πλαίσιο αναλόγως του προβλήματος δοκιμής. Για το σκοπό αυτό μελετήθηκε ένα πλήθος σχετικών δημοσιεύσεων τόσο από τον δημιουργό του εν λόγω αλγορίθμου όσο και από άλλους ερευνητές του πεδίου που επέκτειναν και βελτίωσαν τον αλγόριθμο δημιουργώντας πολυάριθμες παραλλαγές.

1.2 Διάρθρωση Εργασίας

Η δομή της εργασίας ακολουθεί παρακάτω με το θέμα του κάθε κεφαλαίου:

- Κεφάλαιο 1: περιλαμβάνει τον στόχο της εργασίας και την εισαγωγή στο θέμα που πραγματεύεται.
- Κεφάλαιο 2: παρουσιάζεται το απαραίτητο θεωρητικό υπόβαθρο που απαιτείται για την πλήρη κατανόηση των εννοιών που αναφέρονται στην εργασία. Το κεφάλαιο περιλαμβάνει πέντε (5) ενότητες οι οποίες αφορούν την εισαγωγή στην βελτιστοποίηση, τους μεταερευτικούς αλγορίθμους με τις κατηγορίες τους και τα ειδικά τους χαρακτηριστικά, τους εξελικτικούς αλγορίθμους, τους αλγορίθμους νοημοσύνης σμήνους και τέλος τους αλγορίθμους βελτιστοποίησης που βασίζονται στην κοινωνία των λύκων.
- Κεφάλαιο 3: σε αυτό το κεφάλαιο γίνεται αναλυτική παρουσίαση του Grey Wolf Optimizer, αρχικά γίνεται μία εισαγωγή στον αλγόριθμο, ακολούθως περιγράφονται τα χαρακτηριστικά των λύκων στον φυσικό κόσμο τα οποία χρησιμοποιούνται από τον αλγόριθμο για προσομοίωση της εξερεύνησης και της εκμετάλλευσης των πιθανών λύσεων ενώ ακολουθεί η μαθηματική

μοντελοποίηση τους και η παρουσίαση της ενσωμάτωσης τους στον αλγόριθμο, ενώ εν συνεχεία περιγράφονται οι περιορισμοί και τα μειονεκτήματα του. Στην πέμπτη και τελευταία ενότητα (3.5) του κεφαλαίου περιγράφονται συνοπτικά οι παραλλαγές του αλγορίθμου και γίνεται αναφορά στις εφαρμογές του σε προβλήματα του πραγματικού κόσμου.

- Κεφάλαιο 4: αρχικά περιγράφεται η γλώσσα προγραμματισμού και οι λόγοι που αυτή επιλέχθηκε να χρησιμοποιηθεί για την ανάπτυξη του κώδικα, εν συνεχεία περιγράφονται οι βιβλιοθήκες και τα πακέτα της γλώσσας που επιλέχθηκαν καθώς και ο ρόλος που διαδραμάτισαν στην κατά το δυνατόν βέλτιστη υλοποίηση του αλγορίθμου σύμφωνα με την βιβλιογραφία. Στην τρίτη ενότητα περιγράφονται τα λογισμικά και τα εργαλεία που χρησιμοποιήθηκαν ενώ στην τελευταία ενότητα γίνεται περιγραφή της προγραμματιστικής υλοποίησης καθώς με χρήση ψευδοκώδικα παρουσιάζεται η λογική των κυριότερων συναρτήσεων που αναπτύχθηκαν.
- Κεφάλαιο 5: περιγράφει τις συναρτήσεις δοκιμής που επιλέχθηκαν για την αξιολόγηση του προβλήματος και την σύγκριση των αποτελεσμάτων με άλλους αλγορίθμους όπως είναι ο Αλγόριθμος Σμήνους Σωματιδίων – PSO, ο Αλγόριθμος Βαρυτικής Αναζήτησης – GSA και ο Αλγόριθμος Διαφορικής Εξέλιξης – DE.
- Κεφάλαιο 6: αρχικά πραγματοποιείται μία συγκριτική αξιολόγηση του υλοποιημένου GWO με άλλους μεταερευτικούς αλγορίθμους, ακολούθως γίνεται πειραματική μελέτη και μικρορύθμιση (fine-tuning) των παραμέτρων του αλγορίθμου για το δεδομένο σύνολο προβλημάτων και τέλος παρουσιάζονται τα συμπεράσματα που προέκυψαν.
- Παράρτημα Α: Στο παράρτημα Α παρατίθεται ο κώδικας σε python της υλοποίησης μας, επαρκώς σχολιασμένος και με δομημένη τεκμηρίωση των συναρτήσεων του για ευχερέστερη κατανόηση της λειτουργικότητας του.
- Παράρτημα Β: Στο παράρτημα Β παρουσιάζονται τα χαρακτηριστικά του Η/Υ στον οποίο έγιναν οι πειραματικές δοκιμές του GWO.
- Παράρτημα Γ: Στο παράρτημα Γ παρουσιάζεται η Γραφική Διεπαφή Χρήστη (GUI) η οποία αναπτύχθηκε συμπληρωματικά με την υλοποίηση του GWO

σε γλώσσα Python για οπτικοποίηση της διαδικασίας βελτιστοποίησης και ευκολότερης κατανόησης του τρόπου λειτουργίας του Αλγορίθμου.

2. Θεωρητικό Υπόβαθρο

2.1 Εισαγωγή στην Βελτιστοποίηση

Με τον όρο βελτιστοποίηση στον φυσικό κόσμο αναφέρεται η επίτευξη του καλύτερου δυνατού, του άριστου αποτελέσματος [1], έτσι η επίλυση προβλημάτων βελτιστοποίησης αποτελεί την προσπάθεια για την εξεύρεση της βέλτιστης δυνατής λύσης. Αναλόγως του προβλήματος το οποίο καλούμαστε να επιλύσουμε η ανεύρεση της βέλτιστης λύσης αποτελεί ένα δύσκολο πρόβλημα. Η δυσκολία έγκειται στο γεγονός της ύπαρξης πολλών πιθανών λύσεων.

Σε αυτόν τον τομέα τα μαθηματικά δίνουν για μία ακόμα φορά διάφορα εργαλεία ώστε να μας βοηθήσουν στην εξερεύνηση πιθανών λύσεων που σε άλλη περίπτωση θα ήταν υπερβολικά δύσκολο αν όχι αδύνατο να τις ανακαλύψουμε, καθώς και την επιλογή της βέλτιστης εξ αυτών ή έστω μίας αρκούντως ικανοποιητικής λύσης για κάθε δεδομένο πρόβλημα. Μέσω της μαθηματικής μοντελοποίησης τα προβλήματα είναι πλέον δυνατόν να επιλυθούν με τα κατάλληλα εργαλεία που δεν είναι άλλα από διάφορες αλγοριθμικές τεχνικές.

Η μέθοδος της βελτιστοποίησης θα μπορούσαμε να ισχυριστούμε ότι περιλαμβάνει τα κάτωθι πέντε στάδια:

- α) Αναγνώριση, κατηγοριοποίηση και περιγραφή του προβλήματος. Αποτελεί ίσως το πιο σημαντικό στάδιο καθότι ενδεχόμενο λάθος σε αυτό το σημείο θα οδηγήσει μοιραία σε αποτυχία σε επόμενο στάδιο.
- β) Καθορισμός Παραμέτρων και μαθηματική μοντελοποίηση (Formulation – Mathematical Modeling).
- γ) Οριοθέτηση του προβλήματος με βάσει τους περιορισμούς του (Constraints)
- δ) Εξερεύνηση του χώρου αναζήτησης και επιλογή της βέλτιστης λύσης ή έστω μιας αρκούντως ικανοποιητικής λύσης.
- ε) Έλεγχος της επιλεχθείσας λύσης.

Τα τρία πρώτα στάδια αποτελούν την μοντελοποίηση του προβλήματος καθώς μέσω αυτών προκύπτει η επιστημονική περιγραφή του δοθέντος προβλήματος του

πραγματικού κόσμου. Είναι ιδιαίτερα σημαντικό η μοντελοποίηση να γίνει με τον βέλτιστο τρόπο καθώς εξασφαλίζεται έτσι η καταλληλότητα της υπολογισθείσας λύσης.[2]

Συναφώς το μαθηματικό μοντέλο περιλαμβάνει:

- Συνεχείς ή/και διακριτές μεταβλητές, καθώς και λοιπές παραμέτρους, οι οποίες ανάλογα με τις τιμές που θα λάβουν, οδηγούν σε διαφορετική λύση του προβλήματος.
- Περιορισμούς με την μορφή ανισώσεων ή εξισώσεων, οι οποίοι διατυπώνονται με χρήση των δοθεισών μεταβλητών και εκφράζουν συμβάσεις που πρέπει να γίνουν τηρούνται αυστηρά, ώστε να υπολογίζονται αποδεκτές (νόμιμες) λύσεις.
- Την αντικειμενική συνάρτηση (objective function) η οποία καλείται είτε να ελαχιστοποιηθεί είτε να μεγιστοποιηθεί ανάλογα με τη φύση του εκάστοτε προβλήματος.

Ένα από τα πιο γνωστά πρόβλημα στην επιστήμη των υπολογιστών (Computer Science) είναι το πρόβλημα του Περιοδεύοντος Πωλητή (Traveling Salesman Problem – TSP), το οποίο περιγράφηκε για πρώτη φορά από τον Γερμανό μαθηματικό Carl Menger [3] και αποτελεί χαρακτηριστικό πρόβλημα ελαχιστοποίησης.

Παρόμοια προβλήματα απαντώνται και στον πραγματικό κόσμο, όπως η δρομολόγηση πλοίων και αεροσκαφών μέσω διαθέσιμων διαύλων ναυσιπλοΐας και αεροδιαδρόμων αντίστοιχα. Για την επίλυση τους έχουν αξιοποιηθεί, μεταξύ άλλων, εξελικτικοί μεταερευτικοί αλγόριθμοι, όπως ο Γενετικός Αλγόριθμος (Genetic Algorithm) [4].

Συναφές πρόβλημα αποτελεί και η επιλογή βέλτιστης δρομολόγησης μη επανδρωμένων οχημάτων αυτοκτονίας γνωστά και ως καμικάζι drones, με σκοπό την μεγιστοποίηση της πιθανότητας εξουδετέρωσης διαφόρων στόχων στρατιωτικού και μη ενδιαφέροντος. [5], [6] Επιπλέον, ένα άλλο αξιόλογο πεδίο είναι οι οικονομικές επιστήμες, όπου χρησιμοποιούνται αλγόριθμοι τεχνητής νοημοσύνης για την επίλυση σύνθετων προβλημάτων. Σε αυτά περιλαμβάνεται ο υπολογισμός τάσεων τιμών μετοχών [7], καθώς και η βελτιστοποίηση της βιομηχανικής παραγωγής αγαθών (production optimization) [8], όπου σε αυτές τις περιπτώσεις επιδιώκεται η μεγιστοποίηση των τιμών των αντικειμενικών συναρτήσεων.

Στις υποενότητες που ακολουθούν θα γίνει μνεία σε κάποιες τεχνικές της τεχνητής και της υπολογιστικής νοημοσύνης ενώ θα δοθεί έμφαση στους μεταερευνητικούς αλγόριθμους νοημοσύνης σμήνους και πιο συγκεκριμένα θα γίνει εκτενής αναφορά στο κεφάλαιο 2 στον αλγόριθμο Grey Wolf Optimizer [9].

2.2 Μεταερευνητικοί Αλγόριθμοι

Σύμφωνα με όσα αναφέραμε προηγουμένως γίνεται εύκολα αντιληπτό ότι οι εφαρμογές της βελτιστοποίησης είναι αμέτρητες, κάθε στιγμή κάποιοι άνθρωποι επιφορτίζονται με την επίλυση προβλημάτων βελτιστοποίησης, είτε πρόκειται για χρηματιστές σε χρηματιστηριακές εταιρίες, είτε για ερευνητές σε μεγάλες εταιρείες παρασκευής φαρμάκων για δυσίατες μέχρι στιγμής ασθένειες, είτε για μηχανικούς σε εταιρείες αεροδιαστημικής όπως η SPACE-X. Κοινοί παρονομαστές είναι η τάχιστη επίλυση και το χαμηλότερο δυνατό κόστος. Η χρήση προσεγγιστικών αλγορίθμων αποτελεί το εργαλείο προς την επίτευξη αυτού του σκοπού, πιο συγκεκριμένα υπάρχουν δύο κατηγορίες προσεγγιστικών αλγορίθμων, η κατηγορία των ευρετικών αλγορίθμων (specific heuristic) και η κατηγορία των μεταερευνητικών αλγορίθμων (metaheuristics) [10].

Οι ευρετικοί αλγόριθμοι σύμφωνα με τον Pearl [11] αποτελούν κριτήρια, μεθόδους ή αρχές που χρησιμοποιούνται για την επιλογή της πιο κατάλληλης δράσης ανάμεσα σε πολλές εναλλακτικές, με σκοπό την επίτευξη ενός συγκεκριμένου στόχου. Αντιπροσωπεύουν έναν συμβιβασμό μεταξύ δύο απαιτήσεων, από την μία πλευρά, την ανάγκη για απλότητα στις ευρετικές μεθόδους και από την άλλη, την ικανότητα να διακρίνουν τις καλές από τις κακές επιλογές.

Οι μεταερευνητικοί αλγόριθμοι σύμφωνα με τους Award, Gass et al [12] αποτελούν αλγορίθμους υψηλού επιπέδου που δεν εξαρτώνται από το συγκεκριμένο, προς επίλυση, πρόβλημα και παρέχουν ένα σύνολο οδηγιών, κατευθύνσεων και στρατηγικών για την ανάπτυξη ευρετικών αλγορίθμων βελτιστοποίησης. Ο όρος αυτός εισήχθη για πρώτη φορά από τον Glover [13].

Η κύρια διαφορά μεταξύ των δύο προαναφερθέντων κατηγοριών έγκειται στο γεγονός ότι οι ευρετικοί αλγόριθμοι βασίζονται σε μία μόνο υποψήφια λύση, εξερευνούν επαναληπτικά τον χώρο αναζήτησης προκειμένου να εντοπίσουν την βέλτιστη λύση. Αντίθετα, οι μεταερευνητικοί αλγόριθμοι χρησιμοποιούν έναν πληθυσμό λύσεων και εξερευνούν πολλές

περιοχές του χώρου αναζήτησης, με στόχο την εύρεση καλών λύσεων. [14] Στην παρούσα ενότητα θα εστιάσουμε στους μεταερευτικούς αλγορίθμους.

2.2.1 Ορισμός και Κατηγορίες

Ο όρος μεταερευτικός (metaheuristic), ο οποίος προέρχεται από τα αρχαία ελληνικά και σημαίνει "βρίσκω", από το πρώτο συνθετικό, την λέξη μετά (meta), η οποία ερμηνεύεται ως "πάνω" ή "πέρα". Το meta χρησιμοποιείται για να συμπληρώσει ή να προσθέσει κάποια οντότητα. Για παράδειγμα, meta-data σημαίνει δεδομένα για κάποια δεδομένα. Στην περίπτωση μας τον όρο μεταερευτικός (metaheuristic), μπορούμε να τον ερμηνεύσουμε ως έναν ευρετικό για κάποιους άλλους ευρετικούς.

Λόγω της στοχαστικής τους συμπεριφοράς, οι μεταερευτικοί αλγόριθμοι ξεκινούν τη διαδικασία βελτιστοποίησής δημιουργώντας τυχαίες λύσεις. Σε αντίθεση με τις προσεγγίσεις gradient search, δεν απαιτούν τον υπολογισμό της παραγώγου του χώρου αναζήτησης [15]. Ωστόσο, οι μεταερευτικοί δεν εγγυόνται την εύρεση της ολικά βέλτιστης λύσης, καθώς δεν μπορούν να γνωρίζουν πληροφορίες για τις ανεξερεύνητες περιοχές του χώρου αναζήτησης. Κατά συνέπεια, καθώς συγκλίνουμε προς την βέλτιστη λύση αλλά δεν την εντοπίζουμε ακριβώς ο αλγόριθμος συνήθως σταματά την εκτέλεση του λόγω κάποιας συνθήκης που έχει καθοριστεί, όπως ένα όριο στον αριθμό των επαναλήψεων εκτέλεσης του ή μία πολύ μικρή απόκλιση της τιμής της ευρεθείσας λύσης από την ολικά βέλτιστη [16]. Επισημαίνεται ότι αμφότερα τα ανωτέρω κριτήρια χρησιμοποιήθηκαν στην προγραμματιστική υλοποίηση μας για τον τερματισμό της εκτέλεσης του αλγορίθμου.

Οι μεταερευτικοί αλγόριθμοι, με την αρχική τους σημασία αποτελούν μεθόδους επίλυσης οι οποίες δημιουργούν μια αλληλεπίδραση μεταξύ τοπικών διαδικασιών βελτιστοποίησης και στρατηγικών υψηλού επιπέδου ώστε να δημιουργηθεί μια διαδικασία ικανή να απαγκιστρώνεται από τοπικά βέλτιστα (local optima) και να εξερευνά αποδοτικά τον χώρο των λύσεων. [17], [18]

Σε αυτό το σημείο πρέπει να αναφερθεί το θεώρημα «no free lunch (NFL)», σύμφωνα με το οποίο, καμία μεταερευτική μέθοδος δεν υπερέχει των υπολοίπων, καθότι καμία δεν δύναται να επιλύσει το σύνολο των προβλημάτων βελτιστοποίησης τα οποία είναι αναρίθμητα. [19]

Με άλλα λόγια θα μπορούσαμε να πούμε ότι κάποια συγκεκριμένη μεταερευτική μέθοδος μπορεί να είναι αρκετά υποσχόμενη για την επίλυση ενός υποσυνόλου προβλημάτων

αποδίδοντας ικανοποιητικές λύσεις ωστόσο ο μπορεί να μην έχει την ίδια απόδοση σε ένα διαφορετικό υποσύνολο προβλημάτων. Για το λόγο αυτό η αναζήτηση μεταερευτικών μεθόδων αποτελεί ένα ανοικτό ερευνητικό πεδίο το οποίο αποδίδει κάθε χρόνο είτε εντελώς καινούριες μεθόδους είτε βελτιωμένες εκδοχές ήδη υπαρχουσών. [20]

Οι μεταερευτικοί αλγόριθμοι μπορούν να συνοψιστούν σε δύο μείζονες κατηγορίες ήτοι οι αλγόριθμοι βασισμένοι σε μία αποκλειστική λύση (single based solution metaheuristics) γνωστοί και ως αλγόριθμοι τοπικής αναζήτησης όπως είναι ο αλγόριθμος Αναρρίχησης Λόφου (Hill Climbing), η Προσομοιωμένη Ανόπτυση (Simulated Annealing), η Αναζήτηση Tabu (Tabu Search) κ.α και σε αυτούς που εξετάζουν έναν ολόκληρο πληθυσμό λύσεων ταυτόχρονα (population based metaheuristics) όπως είναι οι εξελικτικοί αλγόριθμοι και οι αλγόριθμοι νοημοσύνης σμήνους. [21] Σε επόμενα κεφάλαια θα εστιάσουμε στους αλγορίθμους που λειτουργούν με πληθυσμό λύσεων (population based solution), αρχικά θα αναφερθούμε, εν συντομία, στους εξελικτικούς αλγορίθμους και εν συνεχεία στους αλγορίθμους νοημοσύνης σμήνους.

2.2.2 Χαρακτηριστικά Μεταερευτικών

Οι μεταερευτικοί αλγόριθμοι, λειτουργούν σε δύο φάσεις με την πρώτη να αφορά την εξερεύνηση του χώρου (exploration), και τη δεύτερη να αφορά την εκμετάλλευση (exploitation) της ευρεθείσας λύσης. Η φάση της αναζήτησης αναφέρεται στην κατά το δυνατόν εκτενέστερη διαδικασία εξερεύνησης υποσχόμενων περιοχών του χώρου αναζήτησης. Ο αλγόριθμος θα πρέπει να διαθέτει στοχαστικούς τελεστές προκειμένου να εκτελεί τυχαιοποιημένη και καθολική αναζήτηση (Global Search). Η εξεύρεση της ισορροπίας μεταξύ των δύο αυτών φάσεων θεωρείται ότι αποτελεί πρόκληση λόγω της στοχαστικής φύσης των μεταερευτικών.

Επίσης διαθέτουν τα εξής κοινά χαρακτηριστικά [22]:

- α) Μπορούν να εφαρμοστούν σε μία πληθώρα προβλημάτων βελτιστοποίησης.
- β) Συνήθως υπολογίζουν ικανοποιητικές λύσεις σε λογικό χρόνο επίλυσης.
- γ) Μπορούν να χρησιμοποιηθούν για την επίλυση πολυτροπικών (multimodal) προβλημάτων.
- δ) Συνήθως υλοποιούνται εύκολα.

- ε) Δεν απαιτούν τα προβλήματα βελτιστοποίησης να είναι κυρτά, δηλαδή να διαθέτουν μόνο ένα ολικό βέλτιστο και δεν χρησιμοποιούν παραγώγους.
- στ) Δεν εγγυόνται την εύρεση της βέλτιστης λύσης.
- ζ) Χρειάζονται λεπτομερή παραμετροποίηση για το κάθε πρόβλημα ξεχωριστά.
- η) Τέλος κάποιες φορές απαιτούν αρκετό χρόνο για να συγκλίνουν σε κάποια αρκούντως ικανοποιητική λύση.

2.2.3 Χρονική Πολυπλοκότητα

Η χρονική πολυπλοκότητα (Time Complexity) γνωστή και ως χρονική αποδοτικότητα (Time Efficiency) χρησιμοποιείται για να ποσοτικοποιήσει την ταχύτητα με την οποία λειτουργεί ένας αλγόριθμος. Τα απαιτούμενα βήματα για την ανάλυση της χρονικής πολυπλοκότητας είναι τα κάτωθι [23]:

- α) Καθορισμός των παραμέτρων που υποδεικνύουν το μέγεθος των εισόδων.
- β) Αναγνώριση της βασικής λειτουργίας (πυρήνας) του αλγορίθμου. Η βασική λειτουργία είναι η λειτουργία που έχει την περισσότερη συνεισφορά στον συνολικό χρόνο υπολογισμού του αλγορίθμου.
- γ) Έλεγχος αν ο αριθμός των επαναλήψεων της βασικής λειτουργίας του αλγορίθμου εξαρτάται μόνο από το μέγεθος της εισόδου.
- δ) Τέλος ο υπολογισμός του αθροίσματος που εκφράζει τον συνολικό αριθμό των επαναλήψεων που εκτελέστηκε η βασική λειτουργία του αλγορίθμου.

Συνοψίζοντας σύμφωνα με [22], [24] λόγω της στοχαστικής τους φύσης η χρονική πολυπλοκότητα των μεταερευτικών μεθόδων αν και δύσκολο να υπολογιστεί ακριβώς είναι προσεγγιστικά πολυωνυμική $O(n^k)$ διότι σχετίζεται με τον αριθμό των επαναλήψεων (iterations), την αναπαράσταση και τον πληθυσμό των πιθανών λύσεων.

2.3 Εξελικτικοί Αλγόριθμοι

Πολλές μεγάλες ανακαλύψεις ήταν αποτέλεσμα της βιονικής δηλαδή της εφαρμογής αρχών της βιολογίας και του φυσικού κόσμου γενικότερα στην μελέτη και κατασκευή διαφόρων συστημάτων. Η μελέτη των νυχτερίδων οδήγησε στην κατασκευή των radar, των ψαριών και των θαλάσσιων θηλαστικών στην κατασκευή υποβρυχίων κ.ο.κ.

Η φυσική εξέλιξη των ειδών μπορεί να θεωρηθεί σαν μία διαδικασία εκμάθησης του τρόπου προσαρμογής στο περιβάλλον και βελτιστοποίησης της καταλληλότητας (fitness) των ειδών. Έτσι μπορούμε να υιοθετήσουμε την εν λόγω αρχή δηλαδή την επιβίωση του καλύτερου στον σχεδιασμό και την βελτιστοποίηση αλγορίθμων. [25]

2.3.1 Ορισμός

Οι Εξελικτικοί Αλγόριθμοι (evolutionary algorithms)¹ είναι αλγόριθμοι που διατηρούν έναν πληθυσμό ατόμων τον οποίο εξελίσσουν με βάση κάποιες διαδικασίες που ονομάζονται τελεστές και προσομοιώνουν την φυσική επιλογή, την διασταύρωση και την μετάλλαξη [26] [27] ή σύμφωνα με [25] είναι αλγόριθμοι που εκτελούν βελτιστοποίηση (optimization) ή εκμάθηση διεργασιών (learning tasks) με την ικανότητα να εξελίσσονται. Στην εικόνα 1 απεικονίζεται ο εξελικτικός κύκλος ο οποίος οπτικοποιεί την εφαρμογή των γενετικών τελεστών στην εξέλιξη των λύσεων.



Εικόνα 1: Εξελικτικός Κύκλος

2.3.2 Κύρια Χαρακτηριστικά

Οι εξελικτικοί αλγόριθμοι σύμφωνα με [25] διαθέτουν τα ακόλουθα τρία κύρια χαρακτηριστικά:

- α) Βασίζονται σε πληθυσμό λύσεων (Population based). Διατηρούν μία ομάδα λύσεων οι οποίες καλούνται πληθυσμός (population) με στόχο την βελτιστοποίηση ή την εκμάθηση του προβλήματος με έναν παράλληλο τρόπο.

¹ αναφέρονται και ως εξελικτικός υπολογισμός (evolutionary computation) καθώς και ως εξελικτική νοημοσύνη (evolutionary intelligence)

- β) Είναι προσανατολισμένοι στην καταλληλότητα της υποψήφιας λύσης (Fitness oriented).
- γ) Είναι καθοδηγούμενοι από την Ποικιλία (Variation driven). Τα άτομα θα περάσουν από κάποιες διεργασίες μετάλλαξης για να προσομοιώσουν την γενετική τροποποίηση των γονιδίων, γεγονός το οποίο είναι θεμελιώδους σημασίας κατά την αναζήτηση.

Με βάσει όσα προαναφέρθηκαν οι εξελικτικοί αλγόριθμοι αποτελούν την εφαρμογή της αρχής της επιβίωσης του καλύτερου ωστόσο υπάρχουν και άλλα φαινόμενα στην φύση τα οποία μπορούμε να χρησιμοποιήσουμε για την επίλυση πολυσύνθετων προβλημάτων, ένα από αυτά είναι η συνεργασία τους σμήνους πουλιών κατά την πτήση τους ή μιας αποικίας μυρμηγκιών προσομοιώνοντας διάφορες λειτουργίες όπως της εύρεσης τροφής, της κατασκευής φωλιάς και άλλες δραστηριότητες των έμβιων όντων. Από την μελέτη αυτών των συμπεριφορών, προτάθηκαν αλγόριθμοι που αξιοποιούν το χαρακτηριστικό της συνεργασίας πολλών μελών μίας ομάδας ζώων.

2.4 Νοημοσύνη Σμήνους

2.4.1 Ορισμός

Η Νοημοσύνη Σμήνους σύμφωνα με τους ορισμούς που δίνονται σε [28] [29], αναφέρεται σε συστήματα που διαθέτουν ένα σύνολο μη ευφυών, αυτόνομων πρακτόρων, οι οποίοι παρουσιάζουν την ικανότητα να επιδεικνύουν συλλογική και ευφυή συμπεριφορά. Αυτή η συμπεριφορά εκδηλώνεται μέσω της παραγωγής απρόβλεπτων, αλλά συγκεκριμένων και ταξινομημένων μοτίβων στο εξωτερικό περιβάλλον.

2.4.2 Εφαρμογές και Χαρακτηριστικά

Ο πρώτος αλγόριθμος της κατηγορίας ήταν ο αλγόριθμος βελτιστοποίησης σμήνους σωματιδίων (Particle Swarm Optimization Algorithm – PSO Algorithm) ο οποίος προτάθηκε το 1995 από τους James Kennedy και Russell Eberhart στο Πανεπιστήμιο Purdue.

Ο PSO βασίζεται στην συμπεριφορά ενός σμήνους πουλιών και ενσωματώνει αρχές της εξέλιξης, το σμήνος σωματιδίων διατρέχει έναν προκαθορισμένο χώρο αναζήτησης προκειμένου να υπολογιστούν οι καταλληλότερες των υποψήφιων λύσεων. Κάθε σωματίδιο

χαρακτηρίζεται από τη θέση και την ταχύτητα του, οι οποίες ενημερώνονται δυναμικά κατά τη διάρκεια της διαδικασίας [30] [31] [32].



Εικόνα 2: Σμήνος φλαμίνγκο σε σχηματισμό. (Author: Nikunj Vasoya – own work, CC BY-SA 3.0. <https://creativecommons.org/licenses/by-sa/3.0/deed.en>.)

Όλοι οι αλγόριθμοι που εντάχθηκαν και συνεχίζουν να εντάσσονται στην κατηγορία των αλγορίθμων σμήνους έχουν προκύψει από την παρατήρηση της συμπεριφοράς εντόμων και άλλων ζώων στην φύση.

Σε ένα σμήνος ή μια αγέλη, κάθε μέλος λειτουργεί αυτόνομα αλλά μέσω αυτών των μεμονωμένων ενεργειών προκύπτει μια συλλογική συμπεριφορά. Μέσω της προσομοίωσης αυτών των συμπεριφορών, όπως είναι η εύρεση τροφής, το ζευγάρωμα, η κατασκευή φωλιάς, η εξερεύνηση επιτυγχάνεται η επίλυση πολύπλοκων προβλημάτων.

Ένα ιδιαίτερα ενδιαφέρον παράδειγμα αποτελεί η επίλυση του προβλήματος του Περιοδεύοντος Πωλητή, στο οποίο έχουμε ήδη αναφερθεί. Η προσέγγιση αυτή βασίζεται στην παρατήρηση της συμπεριφοράς των μελισσών. Οι μέλισσες πετούν από λουλούδι σε λουλούδι, εντοπίζοντας με συγκεκριμένο τρόπο και όχι τυχαία τις θέσεις τους, με σκοπό την ελαχιστοποίηση του χρόνου πτήσης και την εξοικονόμηση ενέργειας.

Σε αυτό το πλαίσιο μπορεί να γίνει προσομοίωση της συμπεριφοράς τους με σκοπό την εύρεση του βέλτιστου μονοπατιού για την δρομολόγηση αεροσκαφών, πλοίων και γενικότερα οχημάτων, επιλύοντας το γνωστό πρόβλημα Vehicle Routing Problem (VRP) το οποίο αποτελεί ένα σύνθετο πρόβλημα συνδυαστικής βελτιστοποίησης. [33]

Οι βασικές ιδιότητες των αλγορίθμων σμήνους είναι οι εξής [34]:

- α) Διαμοιρασμός πληροφοριών μεταξύ πολλαπλών πρακτόρων (λύσεων).

- β) Οι πράκτορες διαθέτουν αυτό-οργάνωσης και συν-εξέλιξης.
- γ) Είναι ιδιαίτερα αποδοτικοί λόγω της συνεργαζόμενης μάθησης (co-learning).
- δ) Μπορούν εύκολα να παραλληλιστούν με πρακτικά προβλήματα του πραγματικού κόσμου.

Η Νοημοσύνη Σμήνους αποτελεί επιπροσθέτως και κλάδο της Υπολογιστικής Νοημοσύνης (Computational Intelligence). Εκτός από την ευκολία στην υλοποίησή, διαθέτουν και την ικανότητα να αξιοποιούν την διαθέσιμη μνήμη για την αποθήκευση των τρεχουσών καλύτερων λύσεων. Σε αντίθεση με τους εξελικτικούς αλγορίθμους στους οποίους αναφερθήκαμε στην προηγούμενη ενότητα, φροντίζουν να διατηρούν πληροφορίες σχετικά με το χώρο αναζήτησης του προβλήματος κατά την εκτέλεση των επαναλήψεων, ενώ χρησιμοποιούν και μικρότερο αριθμό τελεστών. Μπορούμε, λοιπόν, να χαρακτηρίσουμε τους αλγορίθμους αυτούς και ως επαναληπτικούς αλγορίθμους στοχαστικής αναζήτησης, όπου οι ευρετικές πληροφορίες διαμοιράζονται προκειμένου να καθοδηγήσουν την αναζήτηση στις επόμενες επαναλήψεις [35].

2.5 Αλγόριθμοι Βελτιστοποίησης Βασισμένοι στους Λύκους

Οι γκρίζοι λύκοι (*Canis lupus*) ή πιο απλά λύκοι είναι κοινωνικά ζώα τα οποία κυνηγούν χρησιμοποιώντας την δύναμη της αγέλης. Η αγέλη αποτελείται συνήθως από ένα κυρίαρχο ζευγάρι και τους απογόνους της. Οι λύκοι προσεγγίζουν αθόρυβα το θήραμα τους ενώ έχουν αναπτύξει μοναδικά χαρακτηριστικά συνεργατικότητας μέσα στην αγέλη, κινούνται σε ομάδα με σχηματισμό χαλαρού ζεύγους και τείνουν να επιτίθενται στο θήραμα ατομικά.

Στο πεδίο της επίλυσης προβλημάτων βελτιστοποίησης έχουν προταθεί αλγόριθμοι που μοντελοποιούν τις εν λόγω συμπεριφορές. Ο πρώτος αλγόριθμος που προτάθηκε από τους Rui Tang et al. [36] είναι ο αλγόριθμος Αναζήτησης των Λύκων (Wolf Search Algorithm – WSA) τον οποίο θα μελετήσουμε στην υποενότητα που ακολουθεί.

2.5.1 Αλγόριθμος Αναζήτησης των Λύκων - WSA

Ο WSA διαθέτει ως δεδομένο τις πιθανότητες διάφορων περιπτώσεων όπου οι λύκοι έρχονται αντιμέτωποι με τους φυσικούς εχθρούς τους. Όταν συμβαίνει κάτι τέτοιο ο λύκος απομακρύνεται διανύοντας μια μεγάλη απόσταση από την τρέχουσα θέση, γεγονός το οποίο

βοηθάει στον να μην παραμένει ο αλγόριθμος σε τοπικά βέλτιστα (local optima). Η κατεύθυνση και η απόσταση που διανύουν οι λύκοι όταν απομακρύνονται από κάποια απειλή είναι τυχαίες.

Κάθε λύκος στον εν λόγω αλγόριθμο διαθέτει μια εμβέλεια μέσα στην οποία μπορεί να ελέγχει με τις αισθήσεις τους για ύπαρξη θηραμάτων ή για πιθανές απειλές, αυτή η εμβέλεια κάλυψης αναφέρεται και ως απόσταση οπτικής επαφής (visual distance). Αυτή η απόσταση οπτικής επαφής χρησιμοποιείται κατά την αναζήτηση τροφής (ολικά βέλτιστη λύση), για ενημέρωση για την θέση των άλλων μελών της αγέλης ώστε να γίνει αναζήτηση νέας καλύτερης θέσης και για σημάδια που καταδεικνύουν την ύπαρξη απειλών στην ευρύτερη περιοχή έτσι ώστε να απομακρυνθούν από αυτές τις απειλές. Μόλις εντοπίσουν την ύπαρξη θηράματος σε κοντινή απόσταση οι λύκοι ξεκινούν την προσέγγιση με ταχύτητα, αθόρυβα και με ιδιαίτερη προσοχή διότι θέλουν να αποφύγουν τον εντοπισμό τους.

Κατά την διάρκεια αναζήτησης τροφής (search mode) και όταν δεν έχει εντοπιστεί ούτε θήραμα ούτε κάποια απειλή εντός οπτικής εμβέλειας (visual range) οι λύκοι κινούνται με Μπραουνιανή Κίνηση (Brownian motion – BM). [34]

Η Μπραουνιανή Κίνηση (ή Brownian Motion) είναι ένα φυσικό φαινόμενο που περιγράφει την τυχαία και ακανόνιστη κίνηση μικροσκοπικών σωματιδίων, τα οποία αιωρούνται σε ρευστό (όπως το νερό). Ονομάστηκε έτσι προς τιμήν του Σκωτσέζου βοτανολόγου Ρόμπερτ Μπράουν, ο οποίος το 1827 παρατήρησε για πρώτη φορά αυτό το φαινόμενο ενώ μελετούσε κόκκους γύρης σε σταγόνες νερού υπο το μικροσκόπιο. Ο Μπράουν παρατήρησε ότι τα σωματίδια φαίνονται να κινούνται συνεχώς και τυχαία, χωρίς να δέχονται κάποια εξωτερική δύναμη. [37]

Η Μπραουνιανή Κίνηση αποτελεί θεμέλιο λίθο στη σύγχρονη θεωρία της Χρηματοοικονομικής, καθώς χρησιμοποιείται για τη μαθηματική μοντελοποίηση της τυχαίας διακύμανσης των τιμών μετοχών και άλλων χρηματοοικονομικών προϊόντων. [38]

Η αναζήτηση της τροφής από τους λύκους διαθέτει τρεις κανόνες οι οποίοι μπορούν να μοντελοποιηθούν μαθηματικά και συνοψίζονται ως εξής:

- α) Κάθε λύκος έχει μία συγκεκριμένη οπτική ακτίνα μέσα στην οποία μπορεί να ανιχνεύει πιθανές νέες θέσεις. Η απόσταση μεταξύ της τρέχουσας θέσης του λύκου και μιας υποψήφιας γειτονικής θέσης υπολογίζεται μέσω της απόστασης Minkowski (Εξίσωση 1.1):

$$d(x_i, x_c) = \left(\sum_{k=1}^n |x_{i,k} - x_{c,k}|^\lambda \right)^{\frac{1}{\lambda}} \quad (1.1)$$

όπου:

- x_i είναι η τρέχουσα θέση του λύκου,
- x_c αντιπροσωπεύει όλες τις δυνατές γειτονικές θέσεις κοντά στο x_i ,
- n είναι η διάσταση του χώρου αναζήτησης,
- λ είναι μία θετική σταθερά (συνήθως $\lambda = 1$ για την απόσταση Manhattan ή $\lambda = 2$ για την Ευκλείδεια απόσταση).

- β) Η καταλληλότητα (fitness) της αντικειμενικής συνάρτησης (objective function) αναπαριστά την ποιότητα της τρέχουσας θέσης του λύκου. Ο λύκος προσπαθεί συνεχώς να μεταβεί σε καλύτερη θέση, ωστόσο αν σε μία θέση από τις υποψήφιες προς μετάβαση υπάρχει κάποιο μέλος της αγέλης τότε επιλέγει να μεταβεί σε αυτή, ανεξαρτήτως κατάταξης της μεταξύ των πιθανών θέσεων, σε κάθε άλλη περίπτωση ο λύκος συνεχίζει να κινείται με Μπραουνιανή κίνηση.
- γ) Υπάρχει η πιθανότητα ο λύκος να διαισθανθεί την παρουσία κάποιας απειλής. Σε αυτή την περίπτωση θα εγκαταλείψει την τρέχουσα θέση του για μία άλλη τυχαία θέση όσο γίνεται πιο μακριά από την απειλή και πέρα από την οπτική ακτίνα του (visual range).

Ο αλγόριθμος WSA διαθέτει τρεις διαφορετικούς τύπους θηρευτικής συμπεριφοράς που λαμβάνουν χώρα διαδοχικά.

- α) **Κυνηγώντας με πρωτοβουλία** (Preying Initiatively): Σε αυτό το στάδιο ο λύκος ελέγχει την περιοχή εντός της οπτικής του ακτίνας για την παρουσία πιθανών θηραμάτων. Εφόσον εντοπιστεί ένα ή περισσότερα, επιλέγει να κινηθεί προς εκείνο που παρουσιάζει την μεγαλύτερη τιμή της αντικειμενικής συνάρτησης (fitness). Κατά τη διαδικασία αυτή, αγνοεί πλήρως τις θέσεις των υπόλοιπων μελών της αγέλης και εστιάζει στο θήραμα που έχει εντοπίσει ο ίδιος και εφόσον δεν υπάρχει άλλο σημείο εντός της οπτικής του εμβέλειας με έχει υψηλότερη τιμή καταλληλότητας, ο λύκος συνεχίζει να κινείται αποκλειστικά προς το συγκεκριμένο θήραμα μέχρι να το προσεγγίσει.

Παρακάτω παρατίθεται η συνάρτηση υπολογισμού της επόμενης θέσης στην κίνηση του λύκου

$$x_i^{t+1} = x_i^t + \beta_0 \cdot e^{-r^2} \cdot (x_j^t - x_i^t) + \text{escape}() \quad (1.2)$$

- Η ποσότητα x_i^t αποτελεί την τρέχουσα θέση του λύκου i κατά την χρονική στιγμή t , καθορίζει την αρχική θέση από την οποία θα γίνουν οι κατάλληλες τροποποιήσεις οι οποίες επηρεάζουν στην κατεύθυνση και το μέγεθος της κίνησης.
 - Η ποσότητα x_j^t αναπαριστά την θέση ενός άλλου λύκου j με καλύτερη τιμή καταλληλότητας, μία καλύτερη θέση ενθαρρύνει τον λύκο να κινηθεί προς αυτήν στοχεύοντας σε υψηλότερη καταλληλότητα.
 - Ο παράγοντας β_0 αποτελεί έναν στοχαστικό παράγοντα και αναπαριστά την επίδραση που έχουν άλλα μέλη της αγέλης σε θέσεις με καλύτερες καταλληλότητες. Ο συγκεκριμένος παράγοντας εξισορροπεί την ορμή της μετάβασης προς καλύτερες λύσεις και την εξερεύνηση του χώρου αναζήτησης. Η κατάλληλη τροποποίηση του β_0 είναι κρίσιμη για την αποτελεσματικότητα του αλγορίθμου καθότι επηρεάζει την ταχύτητα της σύγκλισης του αλγορίθμου και την ικανότητα του να ανακαλύπτει βέλτιστες λύσεις.
 - Η ποσότητα e^{-r^2} είναι μία εκθετική συνάρτηση η οποία μειώνεται όσο αυξάνεται το τετράγωνο της απόστασης r , έτσι αν η απόσταση μεταξύ δύο λύκων είναι μικρή τότε η εν λόγω ποσότητα έχει μία μεγάλη τιμή η οποία ερμηνεύεται ότι η ύπαρξη άλλου λύκου κοντά έχει μεγαλύτερη επίδραση στην κίνηση του λύκου από ότι αν ο άλλος λύκος ήταν μακρύτερα και η μεταβλητή r θα είχε μία μεγάλη τιμή. Η εν λόγω ποσότητα προάγει την αποτελεσματική εξερεύνηση και εκμετάλλευση του χώρου αναζήτησης.
 - Η συνάρτηση $\text{escape}()$ προσθέτει έναν τυχαίο παράγοντα στην κίνηση του λύκου επιτρέποντας έτσι την εξερεύνηση νέων περιοχών, η εν λόγω τακτική βοηθά τον αλγόριθμο να μην εγκλωβιστεί σε τοπικά μέγιστα και επιτρέπει την εξερεύνηση νέων περιοχών του χώρου αναζήτησης. Συνήθως η τυχαιότητα στην μεταπήδηση βρίσκεται εντός κάποιων ορίων ώστε να προληφθεί η περίπτωση εντελώς αυθαίρετων μετακινήσεων οι οποίες μπορεί να είναι αντιπαραγωγικές
- β) **Κυνηγώντας Παθητικά** (Preying Passively): Αν ο λύκος i δεν βρει τροφή ή καλύτερο καταφύγιο που να βρίσκεται και έτερο μέλος της αγέλης τότε ο λύκος i

κυνηγάει παθητικά. Ο λύκος σε αυτή την περίπτωση ελέγχει την γύρω περιοχή εντός της οπτικής του εμβέλειας για εισερχόμενες απειλές και προσπαθεί να βελτιώσει την τρέχουσα θέση του συγκρίνοντας αυτήν με τις θέσεις των υπόλοιπων μελών της αγέλης. Ο συγκεκριμένος υπολογισμός πραγματοποιείται με χρήση της κάτωθι σχέσης 1.3.

$$x_i^{t+1} = x_i^t + a \cdot r \cdot rand() \quad (1.3)$$

Όπου:

- x_i^t είναι η τρέχουσα θέση του λύκου i στη χρονική στιγμή t ,
- a είναι η ταχύτητα του λύκου,
- r είναι η ακτίνα της οπτικής εμβέλειας του λύκου,
- $rand()$ είναι μια συνάρτηση που επιστρέφει τιμές στο διάστημα $[-1,1]$

γ) **Διαφυγή** (escape): Οι λύκοι έχουν πολλούς εχθρούς στο φυσικό περιβάλλον, όταν εντοπιστεί κάποια απειλή ο λύκος θα διαφύγει το ταχύτερο δυνατό και θα εγκατασταθεί σε μία άλλη περιοχή η οποία ήταν εκτός οπτικής εμβέλειας από την θέση που είχε όταν εντόπισε την απειλή. Η εμφάνιση των απειλών μοντελοποιείται μαθηματικά με πιθανότητα η οποία καθορίζεται κατά βούληση από τον χρήστη. Η διαφυγή είναι ένα σημαντικό μέρος του αλγορίθμου WSA γιατί επιτρέπει στον αλγόριθμο να εξερευνά αποτελεσματικότερα και εκτενέστερα τον χώρο αναζήτησης και να μην εγκλωβίζεται σε τοπικά μέγιστα. Η νέα θέση υπολογίζεται από την σχέση 1.4.

$$x_i^{t+1} = x_i^t + a \cdot s \cdot escape() \quad (1.4)$$

Στην ανωτέρω σχέση η μεταβλητή s είναι το μέγεθος του βήματος του λύκου το οποίο πρέπει να είναι πάντα μικρότερο της μεταβλητής r (της οπτικής εμβέλειας του λύκου) και η συνάρτηση $escape()$ είναι μία συνάρτηση την οποία υλοποιεί ο εκάστοτε χρήστης η οποία επιστρέφει μία θέση σε απόσταση μεγαλύτερη της οπτικής εμβέλειας του λύκου καθότι ο λύκος πρέπει να διαφύγει σε θέση την οποία δεν έβλεπε όταν εντόπισε την ύπαρξη απειλής κοντά του και επίσης η θέση αυτή θα πρέπει να είναι το περισσότερο το μισό από την απόσταση μέχρι το όριο του χώρου των λύσεων. [34] [36]

Είσοδος :

r = Ακτίνα της οπτικής εμβέλειας.
 s = Μέγεθος βήματος με το οποίο ένας λύκος αλλάζει τη θέση του κάθε φορά.

α = Συντελεστής ταχύτητας του λύκου.
 p_a = Μία καθοριζόμενη από τον χρήστη πιθανότητα $[0,1]$, που καθορίζει πόσο συχνά εμφανίζεται ένας εχθρός.
 n = Αριθμός λύκων.
 $Itermax$ = Μέγιστος επιτρεπτός αριθμός επαναλήψεων.

Έξοδος:

Βέλτιστη θέση του λύκου και η καταλληλότητα του.

begin

Δημιούργησε τον αρχικό πληθυσμό σε n τυχαίες θέσεις.

while κριτήρια τερματισμού δεν έχουν ικανοποιηθεί **do**

foreach λύκος _{i} **do**

Υπολογίστε τη νέα θέση του λύκου _{i} χρησιμοποιώντας την Μπραουνιανή κίνηση (BM) (εξίσωση 1.3).

αν η νέα θέση του λύκου είναι καλύτερη από την τρέχουσα θέση

Ενημέρωσε την τρέχουσα θέση του λύκου.

Βρές άλλους λύκους στην οπτική εμβέλεια του λύκου _{i}

if δεν υπάρχουν λύκοι **then**

Υπολογίστε τη νέα θέση του λύκου _{i} χρησιμοποιώντας την κίνηση BM (εξίσωση 1.3).

αν η νέα θέση του λύκου είναι καλύτερη από την τρέχουσα θέση του, ενημέρωσε την τρέχουσα θέση του λύκου.

else

Υπολογίστε τη νέα θέση κινώντας τον λύκο προς τον καλύτερο λύκο στην οπτική εμβέλεια (εξίσωση 1.4).

αν η υπολογισμένη νέα θέση του λύκου είναι καλύτερη από την τρέχουσα θέση του, ενημέρωσε την τρέχουσα θέση του λύκου.

end

if $\text{rand}() > p_a$ εκτέλεσε διαφυγή (escape) σε νέα θέση (εξίσωση 1.4) ενημέρωσε την καλύτερη λύση.

end

end

end

Ψευδοκώδικας 1 Αλγόριθμος Αναζήτησης Λύκων (WSA) [34]

2.5.2 Αλγόριθμοι βασισμένοι στον WSA

Στην βιβλιογραφία έχει προταθεί ένας αριθμός αλγορίθμων που είναι εμπνευσμένοι από την διαβίωση των λύκων κάποιοι εκ των οποίων αναφέρονται ακολούθως.

Στον Αλγόριθμο Αγέλης Λύκων – Wolf Pack Algorithm (WPA) ο οποίος προτάθηκε από τους Wu και Zhang [39] υπάρχει πάντα ο επικεφαλής λύκος ο οποίος είναι ο πιο έξυπνος και ο πιο δυναμικός και είναι υπεύθυνος για την αξιολόγηση της τρέχουσας κατάστασης και την συνεκτίμηση των πληροφοριών που λαμβάνει από τους υπόλοιπους λύκους της αγέλης. Ο επικεφαλής λύκος (lead wolf) εντέλει κάποιους επίλεκτους λύκους να κυνηγήσουν στην γύρω περιοχή και να εντοπίσουν την ύπαρξη θηράματος στο πιθανό

πεδίο, αυτοί οι επίλεκτοι λύκοι ονομάζονται ανιχνευτές (scouts). Επίσης υπάρχουν και κάποιοι δυναμικοί λύκοι που ανήκουν στην αγέλη (ferocious wolves)

Οι ανιχνευτές κινούνται στην περιοχή και λαμβάνουν αποφάσεις ανεξάρτητα, αναλόγως της μυρωδιάς θηράματος που εντοπίζουν, η εντονότερη μυρωδιά σημαίνει ότι το θήραμα είναι κοντά.

Μόλις ένας ανιχνευτής λύκος εντοπίσει το ίχνος κάποιου θηράματος θα εκτελέσει το χαρακτηριστικό ουρλιαχτό για να επικοινωνήσει και να αναφέρει σχετικά στον επικεφαλής λύκο. Ο επικεφαλής θα αξιολογήσει την κατάσταση και θα λάβει την απόφαση για τυχούσα κλήση όλων των μελών της αγέλης ώστε να κινηθούν για την περικύκλωση του θηράματος. Αν κληθούν τα μέλη της αγέλης από τον επικεφαλής, τα δυνατότερα μέλη της αγέλης θα κινηθούν τάχιστα προς την κατεύθυνση του ανιχνευτή λύκου.

Ακολούθως, αφού έχουν κατατροπώσει το θήραμα γίνεται ο διαμοιρασμός του, ο οποίος δεν είναι ισομερής αλλά υπάρχει ιεραρχία και ξεκινάει με τους δυνατότερους λύκους και καταλήγει στους ασθενέστερους.

Η συμπεριφορά θήρευσης της αγέλης των λύκων ανάγεται σε τρεις ευφυείς συμπεριφορές, την ανίχνευση, το κάλεσμα μέσω του ουρλιαχτού και την πολιορκητική συμπεριφορά απέναντι στο θήραμα καθώς επίσης και σε δύο ευφυείς κανόνες, τον κανόνα «ο νικητής τα παίρνει όλα» για τον επικεφαλής λύκο και τον κανόνα «ο ισχυρότερος επιβιώνει» για την αγέλη των λύκων.

Αναλυτικότερα:

- α) Κανόνας «ο νικητής τα παίρνει όλα» για τον επικεφαλής λύκο:
Ο λύκος με την μεγαλύτερη καταλληλότητα δηλαδή με την υψηλότερη τιμή της αντικειμενικής συνάρτησης είναι ο επικεφαλής λύκος. Κατά τη διάρκεια κάθε επανάληψης του αλγορίθμου, συγκρίνεται η τιμή της συνάρτησης καταλληλότητας του επικεφαλής λύκου με αυτή των άλλων λύκων. Αν η καταλληλότητα του επικεφαλής λύκου δεν συνεχίζει να είναι η καλύτερη τότε θα αντικατασταθεί.
- β) Συμπεριφορά Ανίχνευσης (Scouting Behavior): Ένας αριθμός από s_num επίλεκτους λύκους, εκτός από τον επικεφαλής λύκο, θεωρούνται ως λύκοι ανίχνευσης (scouters wolves). Αυτοί αναζητούν λύσεις στον προκαθορισμένο χώρο. Y_i είναι η ένταση της μυρωδιάς του θηράματος που

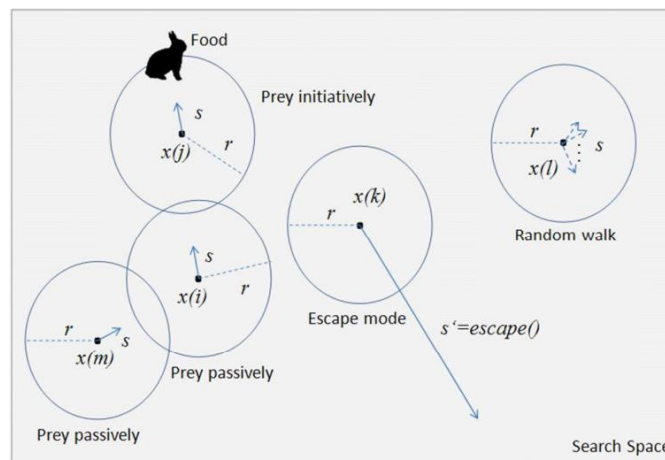
αντιλαμβάνεται ο λύκος ανίχνευσης i και Y_{lead} είναι η ένταση της μυρωδιάς του θηράματος που αντιλαμβάνεται ο επικεφαλής λύκος.

Αν $Y_i > Y_{\text{lead}}$, αυτό σημαίνει ότι ο ανιχνευτής λύκος είναι πιο κοντά στο θήραμα και πιθανώς θα το πιάσει, οπότε ο ανιχνευτής λύκος i γίνεται πλέον ο επικεφαλής λύκος και $Y_{\text{lead}} = Y_i$.

Αν $Y_i < Y_{\text{lead}}$, ο ανιχνευτής λύκος i πραγματοποιεί ένα βήμα προς h διαφορετικές κατευθύνσεις. Το μήκος του βήματος είναι $step_a$. Μετά την κίνηση προς την n -ιοστή κατεύθυνση, η κατάσταση του ανιχνευτή λύκου αναγνώρισης i μοντελοποιείται μαθηματικά με την εξής εξίσωση:

$$x_{id}^n = x_{id} + \sin\left(2\pi + \frac{n}{h}\right) \cdot step_a^d, n = \{1, 2, 3, \dots, h\} \quad (1.5)$$

Πρέπει να σημειωθεί ότι η μεταβλητή h είναι διαφορετική για κάθε λύκο λόγω των διαφορετικών τρόπων αναζήτησης-ανίχνευσης που εκτελούν τα διάφορα μέλη. Η θέση του i -οστού λύκου είναι ένα διάνυσμα $X_i = (x_{i1}, x_{i2}, x_{i3}, \dots, x_{id})$ και η x_{id} είναι d -οστή τιμή του i -οστού λύκου. Η συνάρτηση $Y = f(X)$ αναπαριστά την ένταση της μυρωδιάς του θηράματος η οποία ανιχνεύεται από τον λύκο και αποτελεί και την αντικειμενική συνάρτηση (συνάρτηση καταλληλότητας – fitness function).



Εικόνα 3: Στιγμιότυπο Αλγορίθμου Αναζήτησης Λύκων (WSA).

Επισημαίνεται ότι η ευφυής συμπεριφορά ανίχνευσης θηράματος επισπεύδει την πιθανότητα ο αλγόριθμος να εκτέλεση πλήρη διάσχιση του χώρου των λύσεων. Ο κανόνας «ο νικητή τα παίρνει όλα» του επικεφαλής λύκου και η συμπεριφορά κλήσης των υπόλοιπων μελών επιτρέπουν στους άλλους λύκους να κινηθούν προς τον επικεφαλής, του οποίου η θέση είναι η πιο κοντινή στο θήραμα και είναι και πιο πιθανό να πιάσει το θήραμα.

Επιπροσθέτως βοηθούν τους λύκους να φτάσουν στην περιοχή του ολικού βέλτιστου μόλις μετά από λίγες επαναλήψεις, καθώς το βήμα του αλγορίθμου που περιλαμβάνει την κλήση

των λύκων μέσω του ουρλιαχτού είναι και το πιο μεγάλο για εκτέλεση. Αν το βήμα της κλήσης είναι μικρό τότε το βήμα της πολιορκίας του θηράματος (περικύκλωση του θηράματος – *besieging of the prey*) δίνει στον αλγόριθμο την δυνατότητα να ανακαλύψει νέο χώρο αναζήτησης και να αναζητήσει προσεκτικά το ολικό μέγιστο σε έναν καλό χώρο λύσεων.

Τέλος με την βοήθεια του κανόνα «επιβιώνει ο καταλληλότερος» που αφορά την υπόλοιπη αγέλη ο αλγόριθμος δύναται να αποκτήσει νέους λύκους οι θέσεις των οποίων είναι κοντά στην θέση του καλύτερου λύκου (λύκος επικεφαλής), γεγονός που επιτρέπει σε μεγαλύτερο μήκος του χώρου αναζήτησης να φιλοξενήσει το ολικό μέγιστο ενώ παράλληλα διατηρείται η ποικιλομορφία του πληθυσμού των λύκων σε κάθε επανάληψη. [34] [39]

Ένας ακόμα αλγόριθμος που βασίζεται στην κοινωνία των λύκων είναι ο αλγόριθμος βελτιστοποίησης των Γκρίζων Λύκων (Grey Wolf Optimizer Algorithm) [39]. Ο εν λόγω αλγόριθμος βασίζεται και μοντελοποιεί την ιεραρχία των γκρίζων λύκων εντός της αγέλης, η ιεραρχία αυτή είναι αυστηρή και σαφώς καθορισμένη.

Υπάρχει ένα ζεύγος λύκων (αρσενικό και θηλυκό) τα οποία βρίσκονται στην κορυφή της πυραμίδας, είναι οι επικεφαλείς λύκοι και αποκαλούνται άλφα-alpha. Οι άλφα λύκοι έχουν την ευθύνη να λάβουν όλες τις απαραίτητες αποφάσεις σχετικά με το κυνήγι, τον τόπο ανάπαυσης, την ώρα έγερσης από την ανάπαυση και άλλες. Οι αποφάσεις αυτές επιβάλλονται σε όλα τα υπόλοιπα μέλη της αγέλης.

Στο αμέσως χαμηλότερο επίπεδο της πυραμίδας της ιεραρχίας των γκρίζων λύκων βρίσκονται οι λύκοι βήτα-beta. Οι λύκοι βήτα είναι υποδεέστεροι και συνεπικουρούν τους λύκους άλφα στην λήψη των αποφάσεων (decision-making) ή σε άλλες δραστηριότητες. οι λύκοι βήτα μπορεί να είναι και θηλυκοί και αρσενικοί και αποτελούν την αμέσως καλύτερη επιλογή προς ανάληψη της ηγεσίας και την προαγωγή σε λύκο άλφα σε περίπτωση που κάποιος άλφα λύκος αποβιώσει ή γεράσει και δεν είναι σε θέση να εκτελέσει τα καθήκοντα του.

Στο τελευταίο επίπεδο βρίσκονται οι λύκοι ωμέγα-omega. Οι λύκοι ωμέγα υπακούν πάντα στην εντολές των λύκων των ανώτερων επιπέδων. Είναι οι τελευταίοι λύκοι οι οποίοι θα τραφούν ωστόσο αποτελούν σημαντικό κρίκο της αλυσίδας της αγέλης και χωρίς την ύπαρξή τους δημιουργούνται σοβαρά προβλήματα και ανισορροπία στο εσωτερικό της.

Τέλος οποιασδήποτε λύκος που δεν ανήκει σε ένα από τα προαναφερθέντα επίπεδα ανήκει στο επίπεδο δέλτα-delta το οποίο βρίσκεται κάτω από το βήτα αλλά κυριαρχεί στο ωμέγα. Οι λύκοι αυτούς του επιπέδου είναι οι ανιχνευτές της αγέλης, οι φροντιστές των νεογέννητων λύκων, οι ηλικιωμένοι λύκοι κ.α. Περισσότερα για τον συγκεκριμένο αλγόριθμο αναφέρονται στο επόμενο κεφάλαιο στο οποίο μελετάται εκτενέστερα.

3. Ο Αλγόριθμος Grey Wolf Optimizer

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα πραγματοποιήσουμε μία κατά το δυνατόν εις βάθος μελέτη του αλγορίθμου βελτιστοποίησης των Γκρίζων Λύκων (Grey Wolf Optimizer) ο οποίος προτάθηκε από τους Mirjalili et al [9] το έτος 2014. Ο GWO είναι ένας μεταευρετικός αλγόριθμος βελτιστοποίησης που μοντελοποιεί την συμπεριφορά των γκρίζων λύκων στη φύση.

Οι Αλγόριθμοι Αναζήτησης που βασίζονται σε πληθυσμό λύσεων (population-based) όπως είναι και ο GWO είναι πιο αποτελεσματικοί καθώς είναι σε θέση να πραγματοποιούν την αναζήτηση του βέλτιστου παράλληλα μέσω πολλών πρακτόρων, μειώνοντας έτσι τον απαιτούμενο χρόνο.

Ο GWO μιμείται την ιεραρχία της αγέλης των λύκων και τις τεχνικές κυνηγιού που χρησιμοποιούν. Το κυνήγι, η περικύκλωση και η επίθεση στο θήραμα αποτελούν τις θεμελιώδεις λειτουργίες που χρησιμοποιούνται από τον αλγόριθμο. [40]

3.2 Χαρακτηριστικά των Γκρίζων Λύκων

Ο γκρίζος λύκος ανήκει στην οικογένεια των κυνιδών. Η επιστημονική ονομασία του είδους του είναι *Canis lupus*. Συναντάται κυρίως στη Βόρεια Αμερική και σε ορισμένες περιοχές της Ευρώπης και της Ασίας. Οι λύκοι έχουν χρώμα που κυμαίνεται από λευκό έως γκρι και μαύρο. Στην εικόνα 4 απεικονίζεται μία αγέλη γκρίζων λύκων στην Γαλλία.



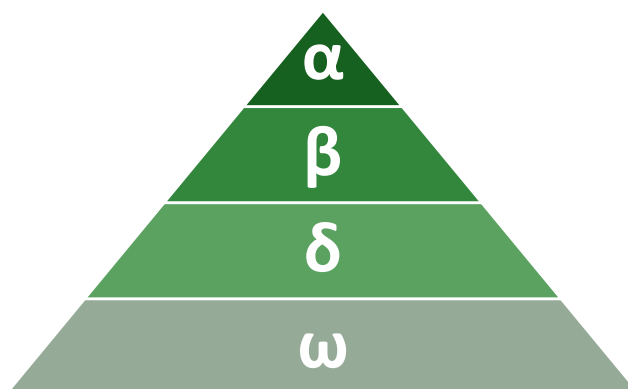
Εικόνα 4: Αγέλη Γκρίζων Λύκων (Author: ALEXANDRE - stock.adobe.com. https://wildark.org/wp-content/uploads/2019/05/AdobeStock_200361935.jpg.)

Οι περιοχές διαβίωσης των λύκων είναι τα δάση, τα βουνά και οι βάλτοι. Οι λύκοι σηματοδοτούν τα όρια των εδαφών τους, τα διεκδικούν και τα υπερασπίζονται. Οι λύκοι έχουν περιοχές που κυμαίνονται από δεκάδες έως εκατοντάδες τετραγωνικά μίλια. Το μέγεθος της περιοχής επηρεάζεται κυρίως από το μέγεθος της αγέλης, τις γειτονικές αγέλες λύκων, το ανθρώπινο περιβάλλον κ.λπ. Οι λύκοι χρησιμοποιούν φωλιές όταν έχουν μικρά. Οι φωλιές δημιουργούνται σε καλά αποστραγγιζόμενο έδαφος, κυρίως κοντά σε νερό σε φυσικές δομές όπως ογκόλιθοι και κορμοί δέντρων. Οι λύκοι τρώνε άλλα ζώα, όπως κουνέλια, ελάφια, σολομούς και τα κουφάρια τους, και μερικές φορές τρέφονται και με βλάστηση.

Οι λύκοι ζουν σε ομάδες των πέντε έως δώδεκα ατόμων, που ονομάζονται αγέλες, αποτελούμενες από έναν πατέρα, μια μητέρα και παιδιά. Ζουν μαζί, κυνηγούν, επικοινωνούν, προστατεύουν την επικράτειά τους και μεγαλώνουν τα παιδιά τους. Οι λύκοι ουρλιάζουν για να επικοινωνήσουν μεταξύ της αγέλης τους καθώς και με τους λύκους άλλων αγελών, πιθανώς για να προειδοποιήσουν για επικείμενο κίνδυνο. Ουρλιάζουν επίσης και για άλλα είδη επικοινωνίας όπως είναι η διεκδίκηση μιας περιοχής, η προειδοποίηση τυχόν εισβολέων της περιοχής τους ή η αναγνώριση άλλων λύκων. Το ουρλιαχτό είναι μοναδικό για μια αγέλη και ουρλιάζουν περισσότερο κατά τη διάρκεια της πανσελήνου. Το ουρλιαχτό ξεκινά από έναν μόνο λύκο και ακολούθως συμμετέχουν και άλλοι.

Οι λύκοι επικοινωνούν όχι μόνο με το ουρλιαχτό, αλλά και με το γρύλισμα, το γάβγισμα, το κλαψούρισμα και άλλα καθώς και μέσω της μυρωδιάς του σώματός τους χρησιμοποιώντας την όσφρηση. Οι λύκοι μπορούν να αντιληφθούν οσμές σε απόσταση μεγαλύτερη τους ενός μιλίου. Η επικοινωνία μεταξύ τους είναι πολύ σημαντική για την επιβίωσή τους, και το μεγαλύτερο μέρος της επικοινωνίας γίνεται μέσω της γλώσσας του σώματος. [40]

Η αγέλη των λύκων διαθέτει αυστηρή ιεραρχία και τα μέλη της είναι σαφώς ενταγμένα σε μία εκ τεσσάρων κατηγοριών. Στην κορυφή της πυραμίδας βρίσκονται οι άλφα λύκοι, ακολουθούν οι βήτα, οι δέλτα και τέλος οι ωμέγα όπως απεικονίζεται στην εικόνα 5.

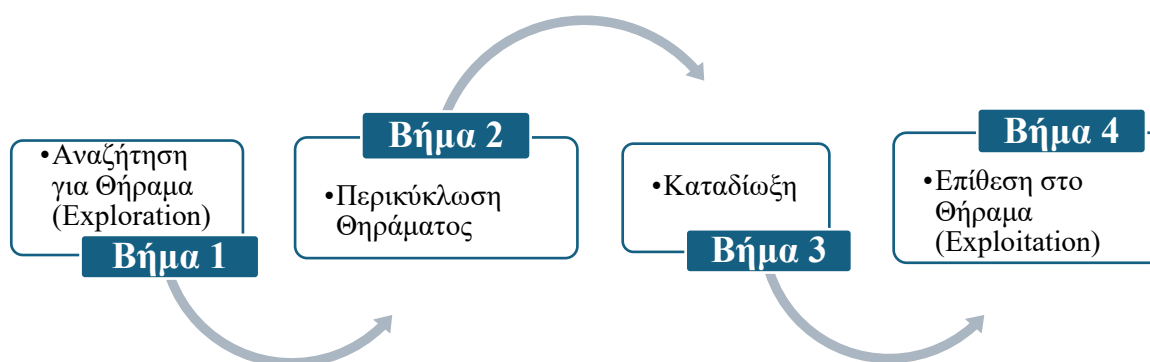


Εικόνα 5: Ιεραρχία Αγέλης Γκρίζων Λύκων.

Σύμφωνα με τους Muro et al [41] εκτός από την αυστηρή δομή στην ιεραρχία της αγέλης οι γκρίζοι λύκοι επιδεικνύουν και μία άλλη ιδιαίτερα ενδιαφέρουσα κοινωνική συμπεριφορά, το ομαδικό κυνήγι το οποίο αποτελείται από τις ακόλουθες φάσεις:

- α) Εντοπισμός και παρακολούθηση του θηράματος (Exploration).
- β) Περικύκλωση του θηράματος.
- γ) Καταδίωξη και Παρενόχληση του θηράματος μέχρι να ακινητοποιηθεί
- δ) Επίθεση στο Θήραμα (Exploitation).

Στην εικόνα 2-3 απεικονίζονται και σχηματικά τα εν λόγω στάδια.



Εικόνα 6: Θηρευτική Διαδικασία Γκρίζων Λύκων [42]

Στην παρακάτω εικόνα (Εικόνα 7) παρουσιάζονται όλα τα στάδια του ομαδικού κυνηγιού ενός βίσωνα στο αμερικανικό εθνικό πάρκο Yellowstone στην πολιτεία Wyoming.



Εικόνα 7: Τυπικές κυνηγετικές συμπεριφορές της αγέλης των λύκων. Ένας βίσωνας στο Εθνικό Πάρκο Yellowstone καταδιώκεται από μια αγέλη εννέα λύκων. (Α) Προσέγγιση, παρακολούθηση και προσέγγιση. (Β-Δ) Καταδίωξη, παρενόχληση και ελιγμός περικύκλωσης. (Ε) [41]

3.3 Μαθηματικό Μοντέλο του Αλγορίθμου

Ο αλγόριθμος των γκριζων λύκων βασίζεται στην ιεραρχία και τις κοινωνικές αλληλεπιδράσεις μεταξύ των λύκων που ανήκουν σε μια αγέλη. Οι τεχνικές κυνηγιού και η πειθαρχημένη συμπεριφορά των γκριζων λύκων αποτελούν την έμπνευση πίσω από τον αλγόριθμο GWO. Οι δραστηριότητες του κυνηγιού, της παρακολούθησης, της περικύκλωσης και της επίθεσης στο θήραμα μοντελοποιούνται μαθηματικά.

Για την αναπαράσταση της ιεραρχίας μέσα στην αγέλη, ορίζουμε τους λύκους των επιπέδων άλφα (α), βήτα (β) και δέλτα (δ) της πυραμίδας της ιεραρχίας ως την πρώτη, δεύτερη και τρίτη καλύτερη λύση αντίστοιχα, ενώ οι υπόλοιπες λύσεις αποτελούν τους λύκους του επιπέδου ωμέγα (ω). Μάλιστα, οι συγκεκριμένες λύσεις παίζουν καθοριστικό ρόλο στην καθοδήγηση της αγέλης κατά τη διαδικασία του κυνηγιού, ενώ οι υπόλοιπες πιθανές λύσεις ανήκουν στη βάση της πυραμίδας, δηλαδή στους ωμέγα (ω). [9]

Οι λύκοι αξιολογούνται και ιεραρχούνται με βάση την καταλληλότητα τους που προκύπτει από κάποια αντικειμενική συνάρτηση που σχετίζεται με το πρόβλημα που επιθυμούμε να επιλύσουμε, ο λύκος με την καλύτερη καταλληλότητα είναι και αυτός που βρίσκεται και πιο κοντά στον στόχο αποτελεί τον λύκο άλφα, ο αμέσως επόμενος λύκος με την καλύτερη καταλληλότητα μετά τον άλφα ονομάζεται λύκος βήτα κ.ο.κ. [43]

3.3.1 Μοντελοποίηση της Περικύκλωσης του Θηράματος

Οι γκρίζοι λύκοι περικυκλώνουν το θήραμα τους κατά την διάρκεια του κυνηγιού, προκειμένου να μοντελοποιηθεί μαθηματικά η εν λόγω διαδικασία έχουν προταθεί οι κάτωθι σχέσεις:

$$\vec{D} = |\vec{C} \cdot \vec{X} - \overline{X(t)}| \quad (2.1)$$

$$\vec{X}(t+1) = \vec{X}_p(t) - \vec{A} \cdot \vec{D} \quad (2.2)$$

Κάτωθι επεξηγούνται οι μεταβλητές των ανωτέρω σχέσεων:

- α) Το t υποδηλώνει τον αριθμό της τρέχουσας επανάληψης στην διαδικασία βελτιστοποίησης.
- β) Το $\vec{X}(t+1)$ είναι διάνυσμα και υποδεικνύει τη νέα θέση του γκρίζου λύκου στην επόμενη χρονική στιγμή (μία υποψήφια λύση).
- γ) Το $\vec{X}_p(t)$ ομοίως είναι διάνυσμα και υποδεικνύει την θέση του θηράματος (η βέλτιστη λύση) την χρονική στιγμή t .
- δ) Τα \vec{A} και \vec{C} είναι διανύσματα συντελεστών που εξυπηρετούν στην προσαρμογή της θέσης των λύκων σε σχέση με το θήραμα, δηλαδή στην θέση των υποψήφιων λύσεων σε σχέση με την βέλτιστη επιθυμητή λύση, έχουν την ίδια διάσταση με τους λύκους (λύσεις) [43] και υπολογίζονται από τις παρακάτω σχέσεις 2.3 και 2.4:

$$\vec{A} = 2 \cdot \vec{a} \cdot \vec{r}_1 - \vec{a} \quad (2.3)$$

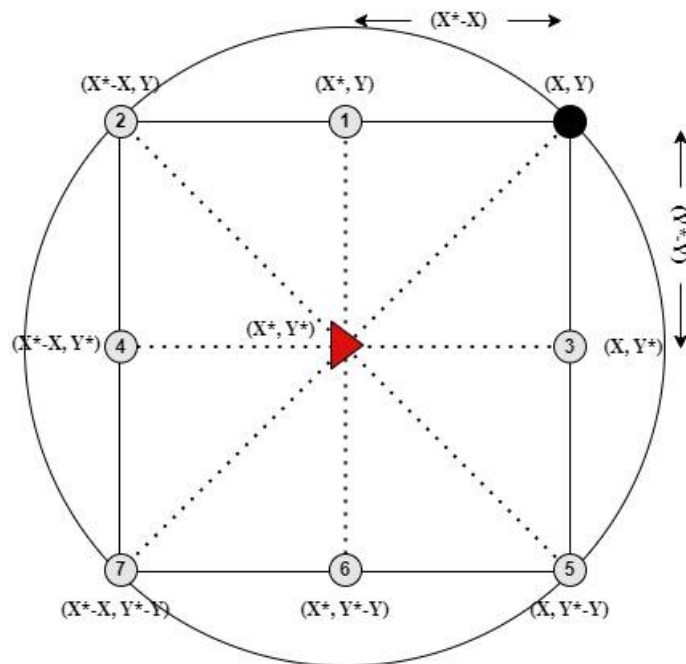
$$\vec{C} = 2 \cdot \vec{r}_2 \quad (2.4)$$

Τα βοηθητικά διανύσματα \vec{a} , \vec{r}_1 και \vec{r}_2 υπολογίζονται ως εξής:

- 1) Τα διανύσματα \vec{r}_1 και \vec{r}_2 είναι τυχαία διανύσματα στο διάστημα $[0,1]$.
- 2) Το διάνυσμα \vec{a} μειώνεται γραμμικά από το 2 έως το 0 κατά τη διάρκεια των επαναλήψεων του αλγορίθμου (διαδικασία αναζήτησης της λύσης).

Στην εικόνα 8 απεικονίζονται οι θέσεις της βέλτιστης λύσης στο κέντρο με κόκκινο χρώμα (θήραμα), της υποψήφιας βέλτιστης λύσης με μαύρο χρώμα (γκρίζος λύκος) και των πιθανών υποψήφιων λύσεων με ανοιχτό γκρι χρώμα.

Όπως φαίνεται και στο σχήμα ο λύκος μπορεί να ενημερώνει την θέση του (X, Y) με βάσει την θέση του θηράματος (X^*, Y^*) και να μετακινείται στο χώρο. Οι θέσεις γύρω από το θήραμα μπορούν να επισκεφθούν από τον λύκο με κατάλληλη τροποποίηση των τιμών των διανυσμάτων \vec{A} και \vec{D} . Επι παραδείγματι για να επισκεφθεί ο λύκος την θέση δύο (2) με συντεταγμένες (X^*-X, Y) τότε αρκεί τα προαναφερθέντα διανύσματα να λάβουν τιμές $\vec{A} = (1, 0)$ και $\vec{D} = (1, 1)$. Τα τυχαία διανύσματα \vec{r}_1 και \vec{r}_2 διασφαλίζουν ότι οι λύκοι της αγέλης δύνανται να καταλάβουν κάθε επιτρεπτή θέση (1-7) όπως αυτές παρουσιάζονται παρακάτω.



Εικόνα 8: Δισδιάστατα Διανύσματα Θέσεων Λύκου και Θηράματος.

Με παρόμοια λογική το δισδιάστατο μοντέλο περικύκλωσης του θηράματος μπορεί να επεκταθεί και σε έναν ν-διάστατο χώρο αναζήτησης, σχηματίζοντας έναν υπερ-κύβο ή μία υπερ-σφαίρα και οι λύκοι θα κινούνται μέσα σε αυτούς γύρω από την βέλτιστη λύση που έχει ανακαλυφθεί μέχρι στιγμής. [9] [40]

3.3.2 Μοντελοποίηση της Αναζήτησης για Θήραμα (Εξερεύνηση)

Κατά την φάση της εξερεύνησης οι γκρίζοι λύκοι αναζητούν τη θέση της βέλτιστης λύσης (θήραμα) βάσει των θέσεων των λύκων άλφα (α), βήτα (β) και δέλτα (δ). Οι λύκοι αποκλίνουν μεταξύ τους μόνο για να ψάξουν για θήραμα και στη συνέχεια συγκλίνουν για να επιτεθούν σε αυτό.

Για να μοντελοποιηθεί μαθηματικά η απόκλιση αυτή, χρησιμοποιούμε το διάνυσμα \vec{A} με τυχαίες τιμές, οι οποίες είναι μεγαλύτερες από 1 και μικρότερες του -1 . Με αυτόν τον τρόπο υποχρεώνουμε τον πράκτορα αναζήτησης να αποκλίνει από το θήραμα διότι όπως προαναφέρθηκε στην προηγούμενη υποενότητα αν οι τιμές του \vec{A} είναι ανάμεσα στο -1 και το 1 τότε πραγματοποιείται σύγκλιση προς τον στόχο (θήραμα).

Ένα άλλο συστατικό του αλγορίθμου που ευνοεί την αναζήτηση είναι το διάνυσμα \vec{C} , το οποίο αντιπροσωπεύει στον φυσικό κόσμο τα τυχόντα εμπόδια που μπορεί να αντιμετωπίσει ο λύκος κατά την προσέγγιση του στην θέση του εντοπισμένου θηράματος. Από μαθηματικής απόψεως μπορεί να γίνει εύκολα αντιληπτό από την σχέση 2.4 ότι το διάνυσμα \vec{C} λαμβάνει τυχαίες τιμές στο διάστημα $[0,2]$, με αυτό τον τρόπο το \vec{C} αποδίδει ένα τυχαίο βάρος σε κάθε θήραμα το οποίο παίζει ρόλο στον υπολογισμό της τρέχουσας απόστασης του λύκου (\vec{D}) από το θήραμα.

Με άλλα λόγια η ύπαρξη εμποδίων προς το στόχο τα οποία δυσκολεύουν και καθυστερούν τον λύκο λαμβάνεται υπόψιν από τον αλγόριθμο και μοντελοποιείται μέσω του \vec{C} το οποίο συνεκτιμάται στον τελικό υπολογισμό της απόστασης του λύκου από το θήραμα η οποία θα επηρεάσει την απόφαση του λύκου για προσέγγιση ή όχι και άρα την απόφαση του πράκτορα αναζήτησης για σύγκλιση ή απόκλιση από την υποψήφια βέλτιστη λύση.

Η εν λόγω πρακτική απόδοσης τιμών στα διανύσματα \vec{A} και \vec{C} προάγει την αναζήτηση στο χώρο επιτρέποντας στον αλγόριθμο να εκτελεί ολική αναζήτηση (global search) αποφεύγοντας την στασιμότητα (stagnation) και την προσκόλληση σε τοπικά βέλτιστες λύσεις (local optima). Αξίζει να σημειωθεί ότι ο αλγόριθμος αποφεύγει τη στασιμότητα σε τοπικά βέλτιστα όχι μόνο στις πρώτες αλλά και στις τελευταίες επαναλήψεις του.

3.3.3 Μοντελοποίηση της Θηρευτικής Διαδικασίας (Κυνήγι)

Σχετικά με τη διαδικασία κυνηγιού, είναι γνωστό πως οι γκρίζοι λύκοι διαθέτουν την ικανότητα να αναγνωρίζουν την τοποθεσία του θηράματος και να το περικυκλώνουν αποτελεσματικά. Την διαδικασία καθοδηγεί συνήθως ο λύκος άλφα (α) ενώ κάποιες φορές οι λύκοι βήτα (β) ή δέλτα (δ) δεν συμμετέχουν. Ωστόσο, σε έναν αφηρημένο χώρο αναζήτησης δεν μπορούμε να γνωρίζουμε τη θέση της βέλτιστης λύσης (θήραμα).

Προκειμένου να γίνει η μαθηματική μοντελοποίηση της θηρευτικής συμπεριφοράς υποθέτουμε ότι ο άλφα (α) λύκος (η καλύτερη υποψήφια λύση), ο βήτα (β) λύκος (η δεύτερη καλύτερη υποψήφια λύση) και ο δέλτα (δ) λύκος (τρίτη καλύτερη υποψήφια λύση) διαθέτουν καλύτερη γνώση για την πιθανή θέση στην οποία βρίσκεται το θήραμα (η βέλτιστη λύση). Για το λόγο αυτό, φροντίζουμε να διατηρούμε στην μνήμη τις τρεις καλύτερες υποψήφιες λύσεις και αναθέτουμε στους υπόλοιπους πράκτορες αναζήτησης (ω) να ενημερώνουν τις θέσεις τους με βάση την θέση των τριών αυτών καλύτερων πρακτόρων (α, β, δ).

Οι κάτωθι τύποι έχουν προταθεί στην βιβλιογραφία για τον σκοπό αυτό:

- α) Οι σχέσεις 2.5 υπολογίζουν την απόσταση του λύκου από το θήραμα δίνονται για κάθε λύκο, αναλόγως του επιπέδου που κατέχει στην πυραμίδα της ιεραρχίας, από τους κάτωθι τύπους, όπου \vec{D} είναι το διάνυσμα της απόστασης του λύκου από το θήραμα.

$$\vec{D}_\alpha = |\vec{C}_1 \cdot \vec{X}_\alpha - \vec{X}|, \quad (2.5)$$

$$\vec{D}_\beta = |\vec{C}_2 \cdot \vec{X}_\beta - \vec{X}|,$$

$$\vec{D}_\delta = |\vec{C}_3 \cdot \vec{X}_\delta - \vec{X}|$$

Αναλυτικότερα οι παράμετροι \vec{D}_α , \vec{D}_β , \vec{D}_δ είναι διανύσματα που δείχνουν την απόσταση σε κάθε διάσταση μεταξύ ενός λύκου από τον άλφα, βήτα και δέλτα λύκο αντίστοιχα. Η παράμετρος \vec{C}_1 είναι διανυσματικός συντελεστής που λαμβάνει τυχαίες τιμές στο διάστημα $[0,2]$, συμβάλλοντας στην ύπαρξη τυχαιότητας στον αλγόριθμο κατά την φάση της αναζήτησης.

- β) Οι σχέσεις 2.6 υπολογίζουν την νέα θέση του κάθε λύκου για τα τρία πρώτα επίπεδα ιεραρχίας σε κάθε επανάληψη:

$$\vec{X}_1 = \vec{X}_\alpha - \vec{A}_1 \cdot (\vec{D}_\alpha), \quad (2.6)$$

$$\vec{X}_2 = \vec{X}_\beta - \vec{A}_2 \cdot (\vec{D}_\beta),$$

$$\vec{X}_3 = \vec{X}_\delta - \vec{A}_3 \cdot (\vec{D}_\delta),$$

Αναλυτικότερα οι μεταβλητές $\vec{X}_1, \vec{X}_2, \vec{X}_3$ είναι διανύσματα των νέων θέσεων κάποιου λύκου με βάσει τις θέσεις των τριών καλύτερων λύκων, υποθέτοντας ότι οι ηγέτες λύκοι έχουν καλύτερες πληροφορίες για την θέση του θηράματος (βέλτιστη λύση). Επίσης συνυπολογίζεται και ο παράγοντας A για την συμπερίληψη τυχαιότητας και ο οποίος θα αναλυθεί εκτενέστερα στην επόμενη υποενότητα (Υποενότητα 3.3.4).

- γ) Η σχέση 2.7 υπολογίζει την νέα θέση των λύκων του τελευταίου επιπέδου των ωμέγα (ω) με βάσει τις θέσεις των τριών λύκων των ανώτερων επιπέδων δεδομένου ότι αποτελούν και τις τρεις καλύτερες υποψήφιες λύσεις.

$$\vec{X}(t+1) = \frac{\vec{X}_1 + \vec{X}_2 + \vec{X}_3}{3}. \quad (2.7)$$

Επισημαίνεται ότι μόνο οι τρεις ανώτεροι πράκτορες αναζήτησης εκτελούν εκτίμηση της θέσης της βέλτιστης λύσης και επαναυπολογισμό της θέσης τους με βάση αυτή ενώ όλοι οι υπόλοιποι πράκτορες αναζήτησης μεταβάλλουν την θέση τους σύμφωνα με τους τρεις καλύτερους βάσει της σχέσης 2.7 χωρίς να πραγματοποιήσουν άλλους υπολογισμούς.

Στην εικόνα 9 παρατηρούμε την τελική θέση του επικεφαλής λύκου άλφα (α) εντός κόκκινου κύκλου, του λύκου βήτα (β) εντός μπλε κύκλου, του λύκου γάμα (γ) εντός κίτρινου κύκλου. Εντός πράσινου κύκλου βρίσκεται κάποιος λύκος ωμέγα (ω) στην αρχική θέση του κατά την τρέχουσα χρονική στιγμή ενώ με χρήση της σχέσης 2.7 που λαμβάνει υπόψιν τις θέσεις των τριών ανώτερων λύκων, αυτή υπολογίζεται και τελικά ο λύκος θα πρέπει να μετακινηθεί εντός του άσπρου κύκλου πλησιέστερα του θηράματος. Ομοίως θα εκτελεστεί ο επαναυπολογισμός της θέσης και για τους υπόλοιπους ωμέγα λύκους. Όπως γίνεται φανερό η τελική θέση των ωμέγα λύκων είναι η μέση θέση που υπολογίζεται από τις θέσεις των άλφα, βήτα και δέλτα λύκων.



Εικόνα 9: Θέσεις λύκων και θηράματος κατά την τελική φάση του κυνηγιού.

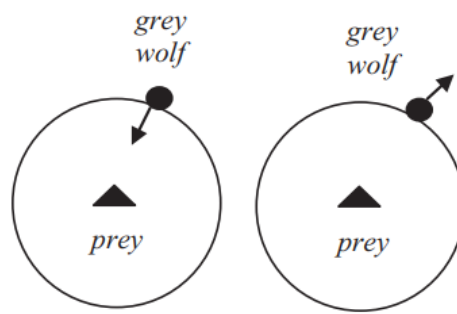
3.3.4 Μοντελοποίηση της Επίθεσης στο Θήραμα (Εκμετάλλευση)

Όπως αναφέρθηκε παραπάνω, οι γκριζοί λύκοι ολοκληρώνουν το κυνήγι επιτιθέμενοι στο θήραμα μόλις αυτό σταματήσει να κινείται. Για να μοντελοποιήσουμε μαθηματικά την προσέγγιση του θηράματος μειώνουμε την τιμή του \vec{a} . Επισημαίνεται ότι το εύρος διακύμανσης του \vec{A} μειώνεται επίσης κατά \vec{a} .

Με άλλα λόγια, το \vec{A} είναι μια τυχαία τιμή στο διάστημα $[2a, 2a]$ όπου το a μειώνεται γραμμικά από το 2 στο 0 κατά τη διάρκεια των επαναλήψεων. Η παράμετρος a μπορεί να υπολογιστεί με χρήση της κάτωθι σχέσης 2.8 [43].

$$a = 2 - \left(\text{iteration number} \cdot \frac{2}{\text{total iterations}} \right). \quad (2.8)$$

Όταν οι τυχαίες τιμές του \vec{A} είναι στο διάστημα $[0,1]$ τότε η επόμενη θέση ενός πράκτορα αναζήτησης (λύκου) μπορεί να είναι σε οποιαδήποτε θέση μεταξύ της τρέχουσας θέσης του και της θέσης του θηράματος. Η εικόνα 10 δείχνει ότι αν το $|A| < 1$ τότε οι λύκοι επιτίθενται στο θήραμα διαφορετικά δεν επιτίθενται και συνεχίζουν να αναζητούν κάποιο άλλο θήραμα.



Εικόνα 10: Επίθεση στο θήραμα (αριστερά) και αναζήτησης για θήραμα (δεξιά). [40]

Με τους τελεστές που έχουν περιγραφεί μέχρι στιγμής, ο αλγόριθμος GWO επιτρέπει στους πράκτορες αναζήτησης να ενημερώνουν τη θέση τους με βάση τη θέση των άλφα, βήτα και δέλτα και να επιτίθενται προς το θήραμα. Ωστόσο, ο αλγόριθμος GWO είναι επιρρεπής σε στασιμότητα και σε εγκλωβισμό σε τοπικές λύσεις. [9]

3.3.5 Σημαντικά Σημεία και Ψευδοκώδικας του Αλγορίθμου

Ακολουθώς αναφέρονται κάποια σημαντικά σημεία του αλγορίθμου:

- Η προτεινόμενη ιεραρχία των λύκων αποτελεί βασικό στοιχείο του αλγορίθμου στην διατήρηση των καλύτερων λύσεων που έχουν ανακαλυφθεί μέχρι στιγμής κατά την εκτέλεση των επαναλήψεων.
- Ο προτεινόμενος μηχανισμός περικύκλωσης του θηράματος όπως περιγράφηκε στην αντίστοιχη υποενότητα καθορίζει μία περιοχή κυκλικού σχήματος γύρω από τις ανευρεθείσες λύσεις, η εν λόγω περιοχή μπορεί να επεκταθεί και στον n -διάστατο χώρο σχηματίζοντας μία υπερσφαίρα.
- Τα τυχαία διανύσματα \vec{A} και \vec{C} συνεισφέρουν στην ύπαρξη διαφορετικής ακτίνας σφαίρας γύρω από κάθε υποψήφια λύση.
- Ο προτεινόμενος θηρευτικός μηχανισμός επιτρέπει στις υποψήφιες λύσεις (λύκοι) να εντοπίζουν την πιθανή θέση της βέλτιστης λύσης (θήραμα).
- Η εξερεύνηση και η εκμετάλλευση διασφαλίζονται από τις προσαρμοστικές τιμές των \vec{a} και \vec{A} οι οποίες επιτρέπουν στον αλγόριθμο να μεταβαίνει ομαλά μεταξύ εξερεύνησης (exploration) και εκμετάλλευσης (exploitation).
- Με τη μείωση του διανύσματος \vec{A} , οι μισές επαναλήψεις αφορούν την εξερεύνηση ($|\vec{A}| \geq 1$) και οι άλλες μισές την εκμετάλλευση ($|\vec{A}| < 1$).
- Ο αλγόριθμος έχει δύο κύριες παραμέτρους προς ρύθμιση (\vec{a} και \vec{C}). [9]

Συναφώς με τα ανωτέρω θα πρέπει να αναφερθούμε και σε κάποιες διαφορές του αλγορίθμου σε σχέση με την συμπεριφορά των λύκων στον φυσικό κόσμο:

- Ο αριθμός των άλφα λύκων στον φυσικό κόσμο είναι πάντοτε δύο ένας αρσενικός και ένας θηλυκός, ο αριθμός των βήτα και των δέλτα λύκων είναι περισσότεροι ωστόσο στον GWO αυτές οι κατηγορίες λύκων αποτελούνται αυστηρά από μόνο ένα άτομο, έτσι ο αλγόριθμος έχει πάντα έναν λύκο άλφα, έναν λύκο βήτα και έναν λύκο δέλτα.
- Στον φυσικό κόσμο η ιεραρχία μέσα στην αγέλη δεν αλλάζει εκτός αν υπάρξει κάποιος θάνατος ή ο άλφα λύκος γεράσει και δεν μπορεί να ανταποκριθεί στα καθήκοντα του, οπότε και μεταπίπτει στην κατηγορία ωμέγα, ωστόσο στον αλγόριθμο σε κάθε επανάληψη γίνεται επαναξιολόγηση της καταλληλότητας των λύκων και η ιεραρχία δύναται να τροποποιηθεί αναλόγως των τιμών των συναρτήσεων καταλληλότητας του συνόλου των λύκων της αγέλης.
- Επίσης σε ένα πραγματικό κυνήγι αγέλης λύκων το θήραμα είναι ορατό από το σύνολο ή μέρος της αγέλης των λύκων, ενώ κατά την διαδικασία της βελτιστοποίησης το θήραμα δεν είναι ορατό και η ακριβής θέση του είναι άγνωστη. Το θήραμα σε πραγματικές καταστάσεις μπορεί να κινηθεί και θα προσπαθήσει να απομακρυνθεί από την αγέλη των λύκων προκειμένου να επιβιώσει, ωστόσο κατά την διαδικασία της βελτιστοποίησης το θήραμα θεωρείται από τον αλγόριθμο ότι βρίσκεται σε συγκεκριμένη θέση, είναι δηλαδή στατικό και το ζητούμενο είναι η ανεύρεση της θέσης του από της αγέλη.
- Στον αλγόριθμο το σύνολο των λύκων της αγέλης κινείται με βάση τις θέσεις των τριών καλύτερων λύκων, ωστόσο στην φύση και οι λύκοι ω μπορούν επίσης να δουν το θήραμα και να εκτελέσουν κινήσεις με βάση την δική τους κρίση.
- Σε πραγματικές συνθήκες δεν μπορούν να είναι ταυτόχρονα στην ίδια θέση στο χώρο δύο λύκοι, ωστόσο στην διαδικασία βελτιστοποίησης μπορούν δύο τεχνητοί λύκοι να κατέχουν ακριβώς την ίδια θέση.

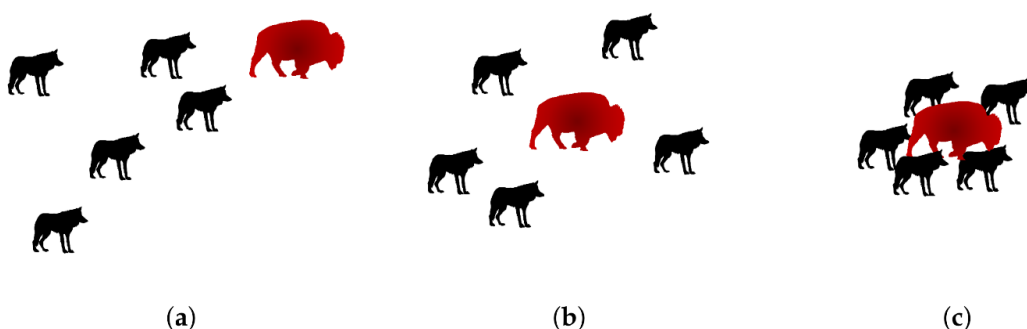
- Το μέγεθος της αγέλης περιορίζεται σε 5 με 12 λύκους στον φυσικό κόσμο, γεγονός που δεν ισχύει στον GWO ο οποίος επιτρέπει και έναν μεγάλο αριθμό λύκων (υποψήφιων λύσεων).
- Οι λύκοι άλφα, βήτα και δέλτα συνήθως είναι κοντά κατά την διαδικασία του πραγματικού κυνηγιού, όμως αυτό μπορεί να μην ισχύει κατά την αναζήτηση του χώρου με τον GWO. [43]

Η λειτουργία του αλγορίθμου συνοψίζεται στα εξής βήματα:

- α) Η αναζήτηση για το θήραμα ξεκινάει από το σχεδιασμό ενός τυχαίου πληθυσμού λύκων που αποτελούν υποψήφιες λύσεις. Θεωρούμε ότι οι απαιτούμενες τιμές εισόδου όπως είναι τα όρια των τιμών των μεταβλητών λαμβάνονται υπόψιν από τον αλγόριθμο με την έναρξη της εκτέλεσης του.
- β) Κατά τη διάρκεια των επαναλήψεων, οι λύκοι εκτιμούν την πιθανή θέση του θηράματος στο χώρο και αναλόγως της σχετικής θέσης τους ως προς αυτό, υπολογίζεται η καταλληλότητα τους αναλόγως της τιμής της αντικειμενικής συνάρτησης και ακολούθως ιεραρχούνται στην πυραμίδα και αναλαμβάνουν τους ρόλους τους. Η κοινωνική ιεραρχία των λύκων όπως προαναφέρθηκε δίνει την δυνατότητα στον αλγόριθμο να διατηρήσει στην μνήμη για κάθε επανάληψη τις τρεις καλύτερες λύσεις που έχουν ανακαλυφθεί μέχρι εκείνη την στιγμή και οι οποίες αντιστοιχούν στη θέση των α , β και δ λύκων, συμβολίζονται με X_α , X_β και X_δ .
- γ) Στην επόμενη φάση κάθε υποψήφια λύση (λύκος), ενημερώνει την θέση της και την απόσταση της σε σχέση με το θήραμα (βέλτιστη λύση), σύμφωνα με τις σχέσεις που αναφέρθηκαν και μοντελοποιούν αυτή την συμπεριφορά.
- δ) Επίσης η παράμετρος \vec{a} μειώνεται γραμμικά σε κάθε επανάληψη από την τιμή 2 στην τελική τιμή 0, ώστε να δοθεί μεγαλύτερη έμφαση στις διαδικασίες της εκμετάλλευσης και της εξερεύνησης αντιστοίχως. Με τη μείωση της παραμέτρου a τροποποιείται και η τιμή των διανυσμάτων συντελεστών \vec{A} και \vec{C} .
- ε) Η μαθηματική μοντελοποίηση της θηρευτικής διαδικασίας δίνει την δυνατότητα στις υποψήφιες λύσεις να εντοπίσουν την θέση της πιθανής

βέλτιστης λύσης και να ενημερώνουν την νέα τους θέση, εφ' όσον αξιολογήσουν ότι η νέα αυτή λύση είναι καλύτερη της προηγούμενης.

- στ) Τέλος, ο αλγόριθμος τερματίζεται όταν ικανοποιηθεί ένα σύνολο από κριτήρια τερματισμού τα οποία έχει καθορίσει ο χρήστης. Τα κριτήρια μπορούν να είναι είτε ένας συγκεκριμένος αριθμός εκτέλεσης επαναλήψεων είτε μία συγκεκριμένη μικρή τιμή στην απόκλιση των καταλληλοτήτων όλων των λύσεων που μοντελοποιεί την συγκέντρωση της αγέλης γύρω από την ίδια θέση.



Εικόνα 11: Διαδικασία ομαδικού κυνηγιού γκρίζων λύκων σε αγέλη (a) εντοπισμός θηράματος, (b) περικύκλωση θηράματος και (c) επίθεση θηράματος. [44]

Ακολούθως παρατίθεται ο Ψευδοκώδικας και το διάγραμμα ροής του αλγορίθμου:

Αρχικοποίησε τον πληθυσμό των γκρίζων λύκων X_i ($i=1,2,...,n$) με τυχαίες θέσεις στο διάστημα $[\alpha, \beta]$, επιλεγμένες με ομοιόμορφη κατανομή.
Αρχικοποίησε τα α , A και C με τυχαίες τιμές στο διάστημα $[\alpha, \beta]$ επιλεγμένες με ομοιόμορφη κατανομή.

Υπολόγισε την καταλληλότητα του κάθε πράκτορα αναζήτησης $f(X_i)$

X_α = 0 καλύτερος πράκτορας αναζήτησης

X_β = ο δεύτερος καλύτερος πράκτορας αναζήτησης

X_γ = ο τρίτος καλύτερος πράκτορας αναζήτησης

$t = 1$ // Αρχικοποίηση του χρόνου (ή αριθμού επαναλήψεων)

while ($t < \text{Μέγιστος Αριθμός Επαναλήψεων}$)

for each πράκτορα αναζήτησης X_i ($i=1,2,...,n$):

 Υπολόγισε τα $|A * X_\alpha - X_i|$, $|A * X_\beta - X_i|$ και $|A * X_\gamma - X_i|$

 Υπολόγισε τα νέα X_i ως εξής:

 // Ανανέωση θέσης με βάση την καλύτερη λύση

$X1 = X_i + A * |X_\alpha - X_i| * \text{rand}()$

 // Ανανέωση θέσης με βάση τη δεύτερη καλύτερη λύση

 ή $X2 = X_i + A * |X_\beta - X_i| * \text{rand}()$

 // Ανανέωση θέσης με βάση την τρίτη καλύτερη λύση

 ή $X3 = X_i + A * |X_\gamma - X_i| * \text{rand}()$

```

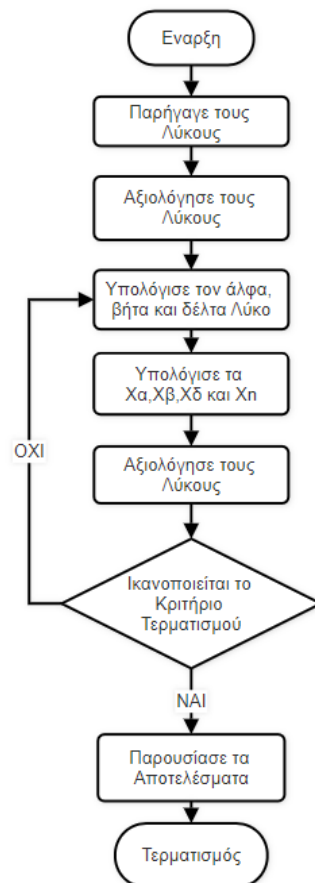
Υπολόγισε τη νέα θέση  $X_i$  με τη μέση τιμή των τριών καλύτερων λύσεων:

$$X_i(t+1) = (X_1 + X_2 + X_3) / 3$$


end for
Ενημέρωσε τα  $\alpha$ ,  $A$  και  $C$ :
 $\alpha = 2 * \text{rand}() - 1$ 
 $A = 2 * \alpha * \text{rand}() - \alpha$ 
 $C = 2 * \text{rand}()$ 
Υπολόγισε τις καταλληλότητες όλων των πρακτόρων αναζήτησης
Ενημέρωσε τα  $X_\alpha$ ,  $X_\beta$  και  $X_\gamma$  με τους καλύτερους πράκτορες αναζήτησης.
// Αύξηση του μετρητή των επαναλήψεων
 $t = t+1$ 
end while
// Επιστροφή τελικής λύση του αλγορίθμου.
Επέστρεψε το  $X_n$ 

```

Ψευδοκώδικας 3 Αλγόριθμος Βελτιστοποίησης Γκρίζων Λύκων (GWO) [9]



Εικόνα 12: Διάγραμμα Ροής του GWO. [43]

3.4 Περιορισμοί και Μειονεκτήματα του Αλγορίθμου

Ο GWO από την στιγμή που προτάθηκε για πρώτη φορά και μέχρι σήμερα έχει χρησιμοποιηθεί ευρέως για την επίλυση πολλών και διαφόρων προβλημάτων σε διάφορους τομείς. Στα θετικά του στοιχεία συμπεριλαμβάνονται η απλότητα του, οι λίγες παράμετροι και συντελεστές ελέγχου και η προσαρμοστική συμπεριφορά του κατά την φάση της αναζήτησης. Τα εν λόγω πλεονεκτήματα είναι και οι βασικοί λόγοι της επιτυχίας του. Ωστόσο και σε πλήρη αντιστοιχία με τους υπόλοιπους αλγορίθμους στοχαστικής βελτιστοποίησης έχει και ορισμένους περιορισμούς και πάσχει αναπόφευκτα από ορισμένα μειονεκτήματα. [45]

Ο βασικός του περιορισμός προέρχεται από το θεώρημα NFL το οποίο έχουμε αναφέρει σε προγενέστερο κεφάλαιο και σύμφωνα με το οποίο κανένας αλγόριθμος βελτιστοποίησης δεν δύναται να λύσει όλα τα προβλήματα. Μοιραία αυτό συνεπάγεται ότι ούτε ο GWO μπορεί και ενδεχομένως να πρέπει να υποστεί κατάλληλες τροποποιήσεις ώστε να επιλύσει τα διάφορα προβλήματα του πραγματικού κόσμου. Ένας ακόμη περιορισμός είναι η μονοσήμαντη φύση αυτού του αλγορίθμου που του επιτρέπει να επιλύει προβλήματα ενός μόνο στόχου, όπως είναι ή μεγιστοποίηση των κερδών ή η ελαχιστοποίηση του κόστους ενός προϊόντος ή μιας υπηρεσίας. Προκειμένου ο GWO να επιλύσει προβλήματα πολλών στόχων, συνδυαστικά προβλήματα όπως είναι το πρόβλημα του περιοδεύοντος πωλητή στο οποίο έχουμε ήδη αναφερθεί προγενέστερα ο αλγόριθμος θα πρέπει να εξοπλιστεί με επιπλέον κατάλληλους τελεστές και μηχανισμό. [45]

Το κύριο μειονέκτημα του GWO είναι η μικρή ικανότητα του να χειρίζεται τις δυσκολίες ενός πολυτροπικού (Multi-Modal) τοπίου αναζήτησης, καθώς φαίνεται ότι και οι τρεις λύκοι άλφα, βήτα και δέλτα τείνουν να συγκλίνουν στην ίδια λύση. Η προσθήκη περισσότερων τυχαίων στοιχείων προκειμένου να επέλθει μετάλλαξη των λύσεων κατά τη βελτιστοποίηση θα αυξήσει την πιθανότητα εύρεσης ενός ολικού βέλτιστου κατά την επίλυση δύσκολων πολυτροπικών προβλημάτων. [45]

Η απόδοση του GWO μειώνεται αισθητά καθώς αυξάνεται ο αριθμός των μεταβλητών, κυρίως λόγω του εγκλωβισμού του αρχικού πληθυσμού σε τοπικά βέλτιστα. Καθώς αυξάνονται οι διαστάσεις του προβλήματος στον χώρο αυξάνεται και η πολυπλοκότητα του χώρου αναζήτησης, με αποτέλεσμα να αυξάνεται και η πιθανότητα πρόωρης σύγκλισης και στασιμότητας σε τοπικά ακρότατα. Αυτή η υποβάθμιση των επιδόσεων υποδηλώνει ότι ο

GWO δυσκολεύεται να διατηρήσει την ισορροπία μεταξύ εξερεύνησης και εκμετάλλευσης. [45]

Επιπροσθέτως οι Niu et al [46] έχουν περιγράψει ένα ελάττωμα του αλγορίθμου αναφορικά με την απόδοση του αναλόγως της φύσεως του προβλήματος βελτιστοποίησης, πιο συγκεκριμένα υπάρχει το ενδεχόμενο ο αλγόριθμος να συμπεριφέρεται άψογα εάν σε ένα συγκεκριμένο πρόβλημα η ολικά βέλτιστη λύση είναι το 0 αλλά να επιδεικνύει πολύ χαμηλή απόδοση όταν η βέλτιστη λύση είναι το 1 ή γενικά αν απέχει επαρκώς από το 0. Αυτή η ασυνέπεια έχει παρατηρηθεί και σε άλλους εξελικτικούς αλγορίθμους. [45], [47]

Επίσης η γρήγορη ταχύτητα σύγκλισης και η επιταχυνόμενη εκμετάλλευση οδηγεί σε στασιμότητα σε τοπικά βέλτιστα λύσεις κατά την επίλυση προβλημάτων με μεγάλο αριθμό μεταβλητών και τοπικών λύσεων. Μια άλλη σημαντική πρόκληση για τον GWO είναι η πιθανή έλλειψη ποικιλομορφίας εντός του πληθυσμού, ιδίως στα μεταγενέστερα στάδια της διαδικασίας βελτιστοποίησης. Καθώς ο αλγόριθμος συγκλίνει προς μια λύση, οι λύκοι μπορεί να μοιάζουν μεταξύ τους, οδηγώντας σε μειωμένη εξερεύνηση του χώρου αναζήτησης. Αυτή η ομοιογένεια μπορεί να εμποδίσει την ικανότητα του αλγορίθμου να ξεφύγει από τα τοπικά βέλτιστα, καθιστώντας δύσκολη την εύρεση του ολικού βέλτιστου. Ένας πληθυσμός λύσεων ικανής ποικιλομορφίας είναι ζωτικής σημασίας για τη διατήρηση της εξερευνητικής δύναμης του αλγορίθμου και την αποφυγή της πρόωρης σύγκλισης. [47]

Τέλος ο αλγόριθμος στερείται σύμφωνα με τους Liu et al [47] αυστηρών θεωρητικών πλαισίων αναφορικά με τις συνθήκες υπό τις οποίες ο αλγόριθμος συγκλίνει σε ένα ολικό βέλτιστο καθώς επίσης δεν υφίστανται και βεβαιώσεις σχετικά με την απόδοση του. Αυτή η αβεβαιότητα η οποία υπάρχει δύναται να εμποδίσει τελείως ή να δυσχεράνει την εφαρμογή του GWO σε κρίσιμα προβλήματα του πραγματικού κόσμου όπου η αξιοπιστία των επιδόσεων είναι ζωτικής σημασίας.

Προκειμένου να ξεπεραστούν οι προαναφερθέντες περιορισμοί και τα μειονεκτήματα της αρχικής έκδοσης του αλγορίθμου έχει προταθεί ένας αριθμός τροποποιημένων εκδόσεων του σε διάφορα σημεία καθώς επίσης και ένας αριθμός από υβριδικές εκδόσεις με άλλους αλγορίθμους. Στην επόμενη ενότητα γίνεται αναφορά ορισμένων εξ αυτών καθώς και ορισμένων πεδίων της επιστήμης στα οποία έχει γίνει εφαρμογή τους.

3.5 Παραλλαγές και Εφαρμογές του Αλγορίθμου

Η πολυπλοκότητα των προβλημάτων βελτιστοποίησης του πραγματικού κόσμου οδήγησε σε προσαρμογές του GWO ώστε να καταστεί συμβατός με τον χώρο αναζήτησης πολύπλοκων πεδίων. Οι αλλαγές στον GWO ομαδοποιούνται συνολικά σε τέσσερις (4) κατηγορίες όπως αναφέρονται παρακάτω:

- α) Τροποποιήσεις στον μηχανισμό ενημέρωσης ώστε να αντιμετωπιστούν διάφορες ελλείψεις και ανεπάρκειες.
- β) Αλλαγές ώστε να ενισχυθούν οι υπάρχοντες τελεστές.
- γ) Δημιουργία μίας υβριδικής έκδοσης ώστε να βελτιωθούν οι λειτουργίες της εξερεύνησης και την εκμετάλλευσης.
- δ) Ανάπτυξη πολλαπλών εκδόσεων για την διαχείριση πλατφορμών παράλληλης επεξεργασίας. [47]

3.5.1 Τροποποιημένες Εκδόσεις του GWO

Αναλόγως του είδους των τροποποιήσεων που έχουν πραγματοποιηθεί στον αρχικό αλγόριθμο προκειμένου αυτός να βελτιωθεί οι τροποποιήσεις ομαδοποιούνται ως εξής:

- α) Σε μηχανισμούς ανανέωσης του αλγορίθμου.
- β) Τροποποιήσεις με εισαγωγή νέων τελεστών.
- γ) Τροποποιήσεις στην ιεραρχία και στην δομή του πληθυσμού.

Στους μηχανισμούς ανανέωσης (refreshing mechanisms) γίνεται προσπάθεια να βελτιωθεί η ισορροπία μεταξύ της εξερεύνησης και την εκμετάλλευσης. Προς αυτή την κατεύθυνση έχει προταθεί από τους Gao και Zhao ο αλγόριθμος GWO με μεταβλητά βάρη (Variable Weights GWO – VW GWO) [48] για να διατηρείται μία αυστηρή κοινωνική ιεραρχία στο εσωτερικό της αγέλης των λύκων. Ο αλγόριθμος απαιτεί το βάρος της θέσης του λύκου άλφα να είναι τουλάχιστον ίσο με το βάρος της θέσης των λύκων βήτα και δέλτα. Μία άλλη έκδοση του αλγορίθμου προτάθηκε από τους Pin Hu et al με καθοδήγηση του άλφα λύκου (Improved Alpha Guided GWO - IAgGWO) [49] στον οποίο έχει γίνει ενσωμάτωση ενός νέου μηχανισμού καθοδήγησης του λύκου άλφα, καθώς επίσης έχει εισαχθεί και ένας τελεστής μετάλλαξης, οι εν λόγω τροποποιήσεις έχουν επιταχύνει την σύγκλιση του αλγορίθμου στο ολικό βέλτιστο, ενώ αποφεύγεται και η στασιμότητα σε τοπικά βέλτιστά.

Μία άλλη βελτιωμένη έκδοση του αλγορίθμου που δίνει έμφαση στην ακριβέστερη μίμηση της ιεραρχίας των λύκων και των τεχνικών τους αναφορικά με την θηρευτική διαδικασία προτάθηκε από τον Luo, με την ονομασία Ενισχυμένος GWO με μοντέλο για τη δυναμική εκτίμηση της θέσης του θηράματος [50], σε αυτή την ενισχυμένη έκδοση κάθε λύκος κινείται απευθείας προς την εκτιμώμενη θέση του θηράματος και η θέση κάθε λύκου αξιολογείται δυναμικά από τον άλφα λύκο. Σύμφωνα με τις δοκιμές που πραγματοποιήθηκαν η συγκεκριμένη έκδοση διαθέτει σαφώς καλύτερη απόδοση σε σχέση με την αρχική έκδοση και άλλες μεταγενέστερες παραλλαγές του. Επιπροσθέτως των προαναφερθέντων παραλλαγών έχουν προταθεί και άλλες όπως είναι ο Random Walk GWO - RWGWO [51] και ο Randomized Balanced Grey Wolf Optimizer – RBGWO [52].

Αναφορικά με τις εκδόσεις του GWO που περιλαμβάνουν νέους τελεστές, όπως ο τελεστής της διασταύρωσης ή η μέθοδος τοπικής αναζήτησης, έχει προταθεί μια έκδοση του αλγορίθμου με χρήση της θεωρίας του χάους (Chaotic GWO). Το χάος αναφέρεται σε ντετερμινιστικά συστήματα που εμφανίζουν τυχαία ή χαοτική συμπεριφορά. [53] Αυτά τα συστήματα είναι μη περιοδικά, δεν συγκλίνουν σε συγκεκριμένες λύσεις, αλλά παραμένουν οριοθετημένα, δηλαδή δεν υπερβαίνουν ορισμένα όρια. Για να εισαχθεί το χάος σε αλγορίθμους βελτιστοποίησης, χρησιμοποιούνται διάφοροι χαοτικοί χάρτες οι οποίοι βασίζονται σε μαθηματικές εξισώσεις που περιγράφουν χαοτικές κινήσεις. Αυτές οι κινήσεις μπορούν να βελτιώσουν τη συμπεριφορά των αλγορίθμων βελτιστοποίησης, καθιστώντας την εξερεύνηση του χώρου αναζήτησης πιο δυναμική και πιο σφαιρική αυξάνοντας την ταχύτητα της σύγκλισης στην ολικά βέλτιστη λύση. [40]

Για να ξεπεραστεί το πρόβλημα της ποικιλομορφίας του πληθυσμού των λύκων και η απόκλιση μεταξύ εξερεύνησης και εκμετάλλευσης προτάθηκε ο Βελτιωμένος GWO (Improved GWO, I-GWO) [54], ενώ ο Ενισχυμένος GWO (Enhanced GWO – EGWO) [55] εισάγει βελτιώσεις στο κυνήγι του θηράματος όπως είναι οι πτήσεις Lévy [56] και η διωνυμική διασταύρωση του πληθυσμού των λύκων.

Τέλος αναφορικά με τις τροποποιημένες εκδόσεις του αλγορίθμου που σχετίζονται με την δομή του πληθυσμού και την κοινωνική ιεραρχία αυτές εστιάζουν στην αύξηση της ποικιλομορφίας των λύσεων και την ενίσχυση της ικανότητας του αλγορίθμου για καθολική αναζήτηση (global search).

Αρχικά προτάθηκε μία εκδοχή του αλγορίθμου στην οποία πέρα από την αγέλη των ενηλίκων λύκων που αναζητά την βέλτιστη λύση, υφίσταται και μία δεύτερη αγέλη παράλληλη στην κύρια αγέλη αποτελούμενοι από νεαρούς λύκους οι οποίοι αλληλοεπιδρούν και μαθαίνουν κοντά στους ενήλικες, η έκδοση αυτή ονομάστηκε Διαδραστικός GWO βασισμένος σε Διδασκαλία-Εκμάθηση με Ενήλικες-Νεαρούς Λύκους (Adult-Pup Teaching–Learning Based Interactive GWO, AP-TBL-IGWO). Η έκδοση αυτή δίνει έμφαση στην ποικιλομορφία των λύσεων, στην βελτίωση της γενίκευσης και στην διαδικασία της αναζήτησης. Οι δύο αγέλες των νεαρών και των ενηλίκων λύκων λειτουργούν ταυτόχρονα και τα μέλη και των δύο αποτελούν πράκτορες αναζήτησης οι οποίοι ερευνούν ταυτόχρονα τον χώρο για την βέλτιστη λύση. Επίσης υπάρχει αλληλεπίδραση και μεταφορά πληροφοριών μεταξύ των νεαρών και των ενηλίκων λύκων. [57]

Μια άλλη εκδοχή με την ονομασία GWO με Ενισχυμένη Ιεραρχία (GWO Enhanced Hierarchy, GWO-EH) δίνει έμφαση στην ιεραρχία των λύκων την οποία τροποποιεί με τέτοιο τρόπο εισάγοντας αυτοπροσδιοριζόμενους συντελεστές βαρύτητας βάσει της καταλληλότητας του κάθε λύκου προκειμένου να γίνει καλύτερη προσομοίωση της ιεραρχίας, ενώ χρησιμοποιείται και μία βελτιωμένη φόρμουλα για τον υπολογισμό της θέσης, έτσι επιτυγχάνεται καλύτερη ταχύτητα σύγκλισης και βελτιωμένη ισορροπία μεταξύ εξερεύνησης και εκμετάλλευσης (exploration-exploitation). [58]

Τέλος από τους Jiang et al προτάθηκε ο Βελτιωμένος GWO με στρατηγική ενισχυμένης ποικιλομορφίας (Diversity Strategy GWO – DSGWO). Σε αυτή την εκδοχή προτείνεται μία διαφορετική ιεραρχία αποτελούμενη από συνολικά έξι (6) λύκους α , β και δ σε αντίθεση με τους τρεις (3) του αρχικού αλγορίθμου, επίσης λειτουργεί και ένας νέος μηχανισμός εξισορρόπησης της εξερεύνησης με την εκμετάλλευση. Οι ανωτέρω τροποποιήσεις στοχεύουν ομοίως στην αύξηση της ποικιλομορφίας των λύσεων και της αποφυγής της στασιμότητας σε τοπικά βέλτιστα. [59]

3.5.2 Υβριδικές Εκδόσεις του GWO

Προκειμένου να αναπτυχθούν καλύτερης ποιότητας λύσεις σε προβλήματα του πραγματικού κόσμου έχει δοθεί έμφαση στην υβριδικότητα υφιστάμενων αλγορίθμων. Έτσι και ο GWO έχει υβριδοποιηθεί με άλλους μεταερευνητικούς αλγορίθμους δημιουργώντας έναν νέο αλγόριθμο που συνδυάζει τα θετικά όλων των επιμέρους αλγορίθμων που έχουν χρησιμοποιηθεί.

Σε αυτό το πλαίσιο αρχικά προτάθηκε ο υβριδικός αλγόριθμος μεταξύ του Αλγορίθμου των Πυροτεχνημάτων (Firework's Algorithm – FWA) και του κλασσικού GWO με την συνδυασμένη ονομασία Firework's GWO (FWGWO). Στον συγκεκριμένο αλγόριθμο γίνεται συνδυασμός της ικανότητας εξερεύνησης του FWA με την ικανότητα εκμετάλλευσης του GWO, δηλαδή συνδυάζονται τα βέλτιστα στοιχεία του κάθε αλγορίθμου. Τα πειραματικά αποτελέσματα απέδειξαν ότι η ταχύτητα σύγκλισης και η ικανότητα καθολικής αναζήτησης εμφάνισαν βελτίωση σε σύγκριση με τους επιμέρους αλγορίθμους. [60]

Επιπροσθέτως μία ακόμα υβριδική έκδοση του αλγορίθμου εστιάζει στην ενσωμάτωση της βιολογικής εξέλιξης με την αρχή «Επιβιώνει ο Καταλληλότερος» - «Survival Of the Fittest (SOF)». Έτσι έχει ενσωματωθεί η χρήση της μεθόδου της Διαφορικής Εξέλιξης (Differential Evolution – DE) προκειμένου η αγέλη των λύκων να ενημερώνεται με βάση την αρχή SOF. Η διαφορική εξέλιξη προσπαθεί επαναληπτικά να βελτιώσει την υποψήφια λύση με βάση κάποιο κριτήριο [61]. Σκοπός των ανωτέρω είναι η αποφυγή της στασιμότητας σε τοπικό ακρότατο. Σε κάθε επανάληψη γίνεται ταξινόμηση των λύκων με βάση την καταλληλότητα που υπολογίζεται και ένας αριθμός από αυτούς με τις χειρότερες καταλληλότητες εξολοθρεύεται, ακολούθως ένας ίδιος αριθμός λύκων παράγονται τυχαία και ενσωματώνονται [62].

Άλλοι αλγόριθμοι που έχουν χρησιμοποιηθεί για την δημιουργία νέων υβριδίων είναι αυτοί της Αναζήτησης του Κούκου (Cuckoo Search) και της Τεχνητής Αποικίας Μελισσών (Artificial Bee Colony) αξιοποιώντας τις τεχνικές του κάθε αλγορίθμου στην αποδοτικότερη εξερεύνηση και ανταλλαγή δεδομένων μεταξύ των πρακτόρων. [63], [64]

Σημειώνεται ότι έχουν προταθεί και αρκετές ακόμη υβριδικές εκδόσεις του αλγορίθμου με χρήση άλλων αλγορίθμων, κοινός παρονομαστής όλων των υβριδίων είναι η αντιμετώπιση των μειονεκτημάτων του βασικού αλγορίθμου, δηλαδή της μειωμένης ποικιλομορφίας των λύσεων και της υψηλής πιθανότητας για γρήγορη σύγκλιση και στασιμότητα σε τοπικό ακρότατο. Οι υβριδικές εκδόσεις εργαλειοποιούν επιμέρους στοιχεία των αλγορίθμων που συμμετέχουν προκειμένου να γίνει βέλτιστη εκμετάλλευση με ενσωμάτωση αυτών στον νέο και ανανεωμένο πλέον αλγόριθμο ο οποίος συνδυάζει τα πλεονεκτήματα των επιμέρους αλγορίθμων.

3.5.3 Στρατηγική Παραλληλισμού (Parallelism)

Στις μεταερευτικές μεθόδους που βασίζονται σε πληθυσμό λύσεων όπως είναι ο GWO, ο παραλληλισμός είναι μια χρήσιμη στρατηγική που επιτρέπει τον διαχωρισμό του πληθυσμού σε υποσύνολα. Μέσω του παραλληλισμού, βελτιώνεται η ποιότητα των λύσεων, ενώ ο χρόνος εκτέλεσης του αλγορίθμου δύναται να μειωθεί σημαντικά.

Προς αυτή την κατεύθυνση προτάθηκε ο αλγόριθμος με ονομασία «Ανακατεμένος κυψελικός GWO - SCEGWO» για τον χρονοπρογραμματισμό εργασιών σε πολλές μηχανές εργοστασίου με στόχο την ελαχιστοποίηση του χρόνου ολοκλήρωσης όλων των εργασιών, το συγκεκριμένο πρόβλημα αποτελεί ένα από τα πιο κλασσικά προβλήματα συνδυαστικής βελτιστοποίησης στον τομέα της επιχειρησιακής έρευνας και της διοικητικής επιστήμης [65].

Λόγω του έντονου ανταγωνισμού στην αγορά όσον αφορά τους συντομότερους κύκλους ζωής των προϊόντων, τη συχνή προσαρμογή των προϊόντων και τις μεταβαλλόμενες απαιτήσεις των πελατών, πολλές σύγχρονες επιχειρήσεις τείνουν να χρησιμοποιούν το ευέλικτο χρονοδιάγραμμα εργασιών (flexible job shop schedule), το οποίο μπορεί να χειριστεί πολλές ροές εργασιών με πολλές πανομοιότυπες μηχανές. Κάθε εργασία αποτελείται από μια ακολουθία διαδοχικών λειτουργιών και κάθε λειτουργία καταλαμβάνει μια χρονική περίοδο, ενώ διεκπεραιώνεται σε μια μηχανή που επιλέγεται από ένα σύνολο υποψήφιων μηχανών. Στην έκδοση του αλγορίθμου του GWO που προτάθηκε χρησιμοποιείται ο τελεστής της μετάλλαξης (crossover) μεταξύ των τριών ανώτερων λύκων, από αυτούς επιλέγεται τυχαία κάθε φορά ένας για να είναι ο ηγέτης των υπόλοιπων πρακτόρων αναζήτησης. Επιπροσθέτως των προαναφερθέντων τροποποιήσεων γίνεται και ενσωμάτωση κυψελικών αυτόματων (cellular automata) με στόχο την αποφυγή της πρόωμης σύγκλισης που εμφανίζεται στον αρχικό GWO [66].

Το Κυψελικό Αυτόματο (Cellular Automaton - CA) είναι ένα διακριτό μοντέλο υπολογισμού που μελετάται στη θεωρία των αυτομάτων. Τα κυψελικά αυτόματα έχουν βρει εφαρμογή σε διάφορους τομείς, όπως η φυσική και η θεωρητική βιολογία. Ένα κυψελικό αυτόματο αποτελείται από ένα πλέγμα κυψελών, η κάθε κυψέλη βρίσκεται σε μία από έναν πεπερασμένο αριθμό καταστάσεων, όπως η ενεργοποίηση και η απενεργοποίηση. Το πλέγμα μπορεί να έχει οποιονδήποτε πεπερασμένο αριθμό διαστάσεων. Για κάθε κυψέλη, ορίζεται ένα σύνολο κυψελών που ονομάζεται γειτονιά. [67]

Έτσι στον προτεινόμενο τροποποιημένο αλγόριθμο GWO με κυψελικά αυτόματα μοντελοποιείται η συμπεριφορά των λύκων στην Καλιφόρνια όπου ο πληθυσμός τους χωρίζεται σε πολλές και ξεχωριστές υπο-ομάδες (υπο-πληθυσμούς) με βάση τη διάταξη της γειτονιάς. Αυτή η ταυτόχρονη προσέγγιση πολλαπλών υπο-πληθυσμών μπορεί να αυξήσει την ποικιλομορφία αναζήτησης και να αντιμετωπίσει τα μειονέκτημα του GWO αναφορικά με την εμφάνιση της πρόωμης σύγκλισης. [47]

3.5.4 Multi-objective GWO (MOGWO)

Ο αλγόριθμος GWO έχει επεκταθεί για την εύρεση του βέλτιστου μετώπου Pareto για προβλήματα πολλαπλών στόχων [68]. Οι δύο πιθανοί τρόποι επίλυσης προβλημάτων πολλαπλών στόχων είναι ο συνδυασμός όλων των πολλαπλών στόχων σε έναν ενιαίο στόχο με κατάλληλα βάρη για τους επιμέρους στόχους ή η εύρεση του συνόλου των μη κυριαρχούμενων λύσεων για όλους τους πολλαπλούς στόχους του προβλήματος.

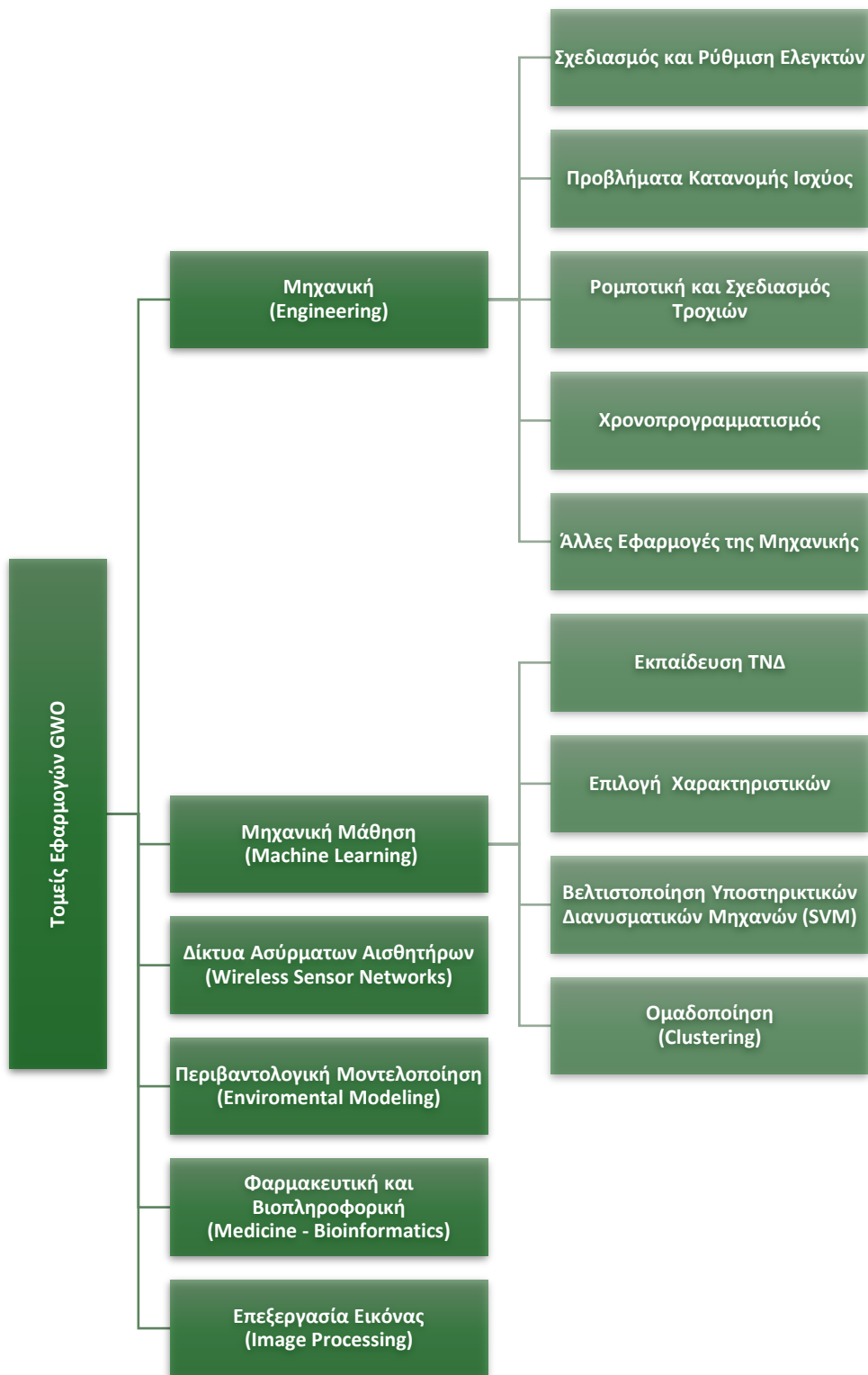
Ο αρχικός GWO έχει τροποποιηθεί ώστε να περιλαμβάνει ένα αρχείο που να περιέχει το σύνολο όλων των μη κυριαρχούμενων λύσεων που προέκυψαν από την εκτέλεση του αλγορίθμου μέχρι εκείνη τη χρονική στιγμή. Οι αρχηγοί του κυνηγιού, α , β και δ λύκοι επιλέγονται από το αρχείο. Το αρχείο έχει ένα μέγιστο όριο στον αριθμό των στοιχείων που μπορεί να αποθηκεύσει. Σε κάθε επανάληψη, εάν μια νέα λύση κυριαρχείται από τουλάχιστον μία εγγραφή του αρχείου τότε δεν επιτρέπεται να αποθηκευτεί στο αρχείο. Εάν η νέα λύση κυριαρχεί σε ένα ή περισσότερα στοιχεία του αρχείου, τα υπάρχοντα στοιχεία απορρίπτονται και το νέο στοιχείο αποθηκεύεται στο αρχείο. Εάν δεν υπάρχει κυριαρχία μεταξύ της υπάρχουσας και της νέας λύσης, η νέα λύση προστίθεται στο αρχείο.

Στον GWO, οι τρεις καλύτερες λύσεις είναι οι λύκοι α , β και δ , οι λεγόμενοι ηγέτες της αγέλης. Στον MOGWO, οι αρχηγοί επιλέγονται από τα λιγότερο πολυσύχναστα τμήματα του χώρου των λύσεων με βάση έναν μηχανισμό ρουλέτας. Εάν υπάρχουν λιγότερα από τρία μέλη στον λιγότερο συνωστισμένο τμήμα του χώρου, οι λύκοι α , β και δ επιλέγονται από άλλα τμήματα με σειρά φθίνοντος συνωστισμού. [40], [68]

3.5.5 Εφαρμογές των Παραλλαγών του GWO

Τα τεράστια πλεονεκτήματα του GWO έχουν οδηγήσει σε σημαντικές εφαρμογές σε πολλούς βασικούς κλάδους. Οι εφαρμογές αυτές μπορούν να χωριστούν σύμφωνα με τους Liu et al [47] στους παρακάτω τομείς:

- α) Της Μηχανικής Μάθησης (Machine Learning – ML) όπου ο GWO έχει χρησιμοποιηθεί για την εκπαίδευση τεχνητών νευρωνικών δικτύων.
- β) Της Μηχανικής (Engineering) όπου έχει χρησιμοποιηθεί για την επίλυση προβλημάτων συνδυαστικής βελτιστοποίησης όπως είναι ο βέλτιστος σχεδιασμός πτερύγων και άλλων κατασκευών.
- γ) Των Δικτύων Ασύρματων Αισθητήρων (Wireless Sensor Networks) με την βελτιστοποίηση της κάλυψης και της ενεργειακής τους απόδοσης.
- δ) Της Περιβαλλοντικής Μοντελοποίησης (environmental modeling) για την βέλτιστη διαχείριση φυσικών πόρων όπως είναι το νερό και η ενέργεια.
- ε) Της φαρμακευτικής (medicine) με την πρόβλεψη αλληλεπιδράσεων μεταξύ φαρμάκων και την προσαρμογή των μοντέλων διάγνωσης ασθενειών.
- στ) Της βιοπληροφορικής (bioinformatics) με την ανάλυση του DNA.
- ζ) Της Επεξεργασίας Εικόνας με την βελτιστοποίηση των αλγορίθμων αποκατάστασης, βελτίωσης ανάλυσης και τμηματοποίησης εικόνων σε εφαρμογές ιατρικής απεικόνισης όπως είναι η ανάλυση μαγνητικών και αξονικών τομογραφιών για την ανίχνευση καρκινικών όγκων. Στην παρακάτω εικόνα (Εικόνα 13) παρουσιάζονται οι κύριοι τομείς και κάποιοι υποτομείς στους οποίους βρίσκει εφαρμογή ο GWO.



Εικόνα 13: Διάγραμμα Εφαρμογών του GWO. [47]

4. Σχεδιασμός – Υλοποίηση του Αλγορίθμου

4.1 Σχετικά με τη Γλώσσα Υλοποίησης

Η Python είναι μία διερμηνευόμενη (interpreted) γλώσσα προγραμματισμού γενικού σκοπού και υψηλού επιπέδου και ανήκει στις γλώσσες αντικειμενοστραφούς προγραμματισμού (*object-oriented programming*), ενώ υποστηρίζει και τον διαδικαστικό (*procedural programming*).

Ο κύριος στόχος της είναι η αναγνωσιμότητα του κώδικά και η ευκολία χρήσης της. Το συντακτικό της python δίνει την δυνατότητα στους προγραμματιστές να συντάξουν προγράμματα με λιγότερες γραμμές κώδικα από ό,τι θα ήταν δυνατόν σε παλαιότερες γλώσσες όπως η C/C++ , η Java και άλλες. Ένα ιδιαίτερο συστατικό της γλώσσας είναι η ύπαρξη πολλών βιβλιοθηκών έτοιμου κώδικα. Επίσης οι διερμηνευτές της Python είναι διαθέσιμοι για εγκατάσταση σε πολλά λειτουργικά συστήματα, επιτρέποντας την εκτέλεση κώδικα σε ευρεία γκάμα συστημάτων. Ωστόσο ένα σημαντικό μειονέκτημα είναι η μικρότερη ταχύτητα εκτέλεσης σε σχέση με τις μεταγλωττιζόμενες (compiled) γλώσσες όπως η C και η C++.

Επισημαίνεται ότι η Python αναπτύσσεται ως ανοιχτό λογισμικό (open source) και η διαχείρισή της γίνεται από τον μη κερδοσκοπικό οργανισμό Python Software Foundation. Ο κώδικας διανέμεται με την άδεια Python Software Foundation License η οποία είναι συμβατή με την άδεια GPL (General Public License) κάνοντάς την ευέλικτη επιλογή για την ανάπτυξη ανοιχτού λογισμικού.

Η γλώσσα δημιουργήθηκε από τον Ολλανδό Guido van Rossum στο ερευνητικό κέντρο Centrum Wiskunde & Informatica (CWI) το οποίο επιτελεί βασική έρευνα στο πεδίο των Μαθηματικών και της Επιστήμης Υπολογιστών. Η Python θεωρείται διάδοχος της γλώσσας προγραμματισμού ABC η οποία αποτέλεσε και την βασική πηγή έμπνευσης.

Η πρώτη έκδοση κυκλοφόρησε το 1991 και από τότε εξελίσσεται συνεχώς με την ανάπτυξη νέων εκδόσεων βάσει των Python Enhancement Proposals ("PEPs"), τα οποία αποτελούν τυποποιημένα κείμενα που περιέχουν προτάσεις για ενσωμάτωση νέων χαρακτηριστικών στην γλώσσα. Η πιο πρόσφατη έκδοση την στιγμή που συντασσόταν η παρούσα εργασία ήταν η 3.13.5 η οποία δημοσιεύτηκε στις 11 Ιουνίου 2025. [69], [70], [71]

Στην παρούσα εργασία χρησιμοποιήθηκε η έκδοση 3.12.11, η οποία είναι και η τελευταία συμβατή έκδοση με τις βιβλιοθήκες και τα πακέτα που θα αναφερθούν ακολούθως.

4.2 Βιβλιοθήκες και πακέτα

4.2.1 Η Βιβλιοθήκη NumPy

Η βιβλιοθήκη NumPy είναι μία ανοικτού κώδικα βιβλιοθήκη της Python η οποία χρησιμοποιείται ευρέως από επιστήμονες και μηχανικούς. Η βιβλιοθήκη περιέχει δομές δεδομένων που αναπαριστούν πολυδιάστατους πίνακες καθώς και μία πληθώρα συναρτήσεων για την εκτέλεση διαφόρων υπολογισμών σε δεδομένα πινάκων. Οι πίνακες NumPy έχουν σταθερό μήκος, σε αντίθεση με τις λίστες οι οποίες, αν απαιτηθεί, αυξάνονται δυναμικά.

Τα στοιχεία ενός πίνακα NumPy απαιτείται να είναι όλα του ιδίου τύπου δεδομένων και επομένως έχουν και το ίδιο μέγεθος στην μνήμη, εξαίρεση αποτελεί η περίπτωση ο πίνακας NumPy να έχει ως στοιχεία αντικείμενα.

Οι πίνακες NumPy παρέχουν το πλεονέκτημα να διευκολύνουν αρκετά την εκτέλεση μαθηματικών και άλλου είδους πράξεων σε μεγάλο πλήθος δεδομένων. Συνήθως οι πράξεις αυτές με την χρήση NumPy εκτελούνται αποδοτικότερα και απαιτούν την συγγραφή λιγότερου κώδικα από ότι χρειάζεται εάν χρησιμοποιηθούν οι ενσωματωμένες συναρτήσεις της Python. [72], [73]

Στον αλγόριθμο μας θα χρησιμοποιηθούν οι εν λόγω πίνακες και οι μέθοδοι που παρέχει η βιβλιοθήκη για την αναπαράσταση των θέσεων των υποψήφιων λύσεων σε n -διάστατα προβλήματα και την εκτέλεση πράξεων επι αυτών.

4.2.2 Η Βιβλιοθήκη SciPy και το Πακέτο SciPy.stats

Ομοίως και η Βιβλιοθήκη SciPy είναι μια βιβλιοθήκη της Python που βασίζεται στη βιβλιοθήκη NumPy. Η SciPy είναι ένα λογισμικό ανοικτού κώδικα για τα μαθηματικά, τις επιστήμες και τη μηχανική. Περιλαμβάνει ενότητες για στατιστική, βελτιστοποίηση, ολοκλήρωση, γραμμική άλγεβρα, μετασχηματισμούς Fourier, επεξεργασία σήματος και εικόνας, κ.α. Το πακέτο scipy.stats περιέχει εργαλεία για στατιστικές αναλύσεις, όπως διάφορες κατανομές (κανονική, Poisson, κ.λπ.), στατιστικούς ελέγχους και τυχαίους αριθμούς από συγκεκριμένες κατανομές. [74]

Στην παρούσα εργασία θα χρησιμοποιήσουμε το πακέτο SciPy.stats για την παραγωγή τυχαίων αριθμών με μεγαλύτερη ακρίβεια ώστε να προσδώσουμε κατά το δυνατόν περισσότερη στοχαστικότητα στον αλγόριθμο μας.

4.2.3 Η Βιβλιοθήκη Matplotlib

Η βιβλιοθήκη Matplotlib αποτελεί μία ολοκληρωμένη βιβλιοθήκη για την δημιουργία στατικών, κινούμενων και διαδραστικών γραφικών με την Python και θα αποτελέσει την βάση πάνω στην οποία θα στηριχτούμε στο πλαίσιο της παρούσας εργασίας για την κατασκευή των απαραίτητων γραφημάτων.

Η Matplotlib προσφέρεται για την δημιουργία ποιοτικών γραφημάτων για την συμπερίληψη τους σε επιστημονικές δημοσιεύσεις, είναι ιδανική για την δημιουργία διαδραστικών γραφικών τα οποία μπορούν να ενημερώνονται να μεγεθύνονται και να επιτελούν μία σειρά από λειτουργίες απεικόνισης. Επίσης τα γραφικά που κατασκευάζονται μπορούν να αποθηκευτούν σε μία πληθώρα τύπων αρχείων γεγονός που προσδίδει αρκετή ευκολία στην μετέπειτα διαχείριση τους. Επίσης η Matplotlib έρχεται ενσωματωμένη στο JupyterLab τον απόγονο του Jupyter Notebook καθώς και σε άλλα γραφικά περιβάλλοντα. Αξιοσημείωτο είναι και το γεγονός ότι περιλαμβάνει πολλά πακέτα από ανεξάρτητους φορείς τα οποία την συμπληρώνουν και την επεκτείνουν, ένα τέτοιο πακέτο είναι η Seaborn. [75]

Η εν λόγω βιβλιοθήκη θα χρησιμοποιηθεί για την απεικόνιση διαφόρων διαγραμμάτων όπως είναι το διάγραμμα της εξέλιξης της θέσης των λύκων (Search Agent Position Plot). Στο διάγραμμα της εικόνας 14 το οποίο κατασκευάστηκε με το κάτωθι τμήμα κώδικα σε python με χρήση του λογισμικού Jupiter² του περιβάλλοντος Anaconda³ απεικονίζεται μια τυχαία διασπορά των πρακτόρων αναζήτησης στον επίπεδο μετά από 100 επαναλήψεις.

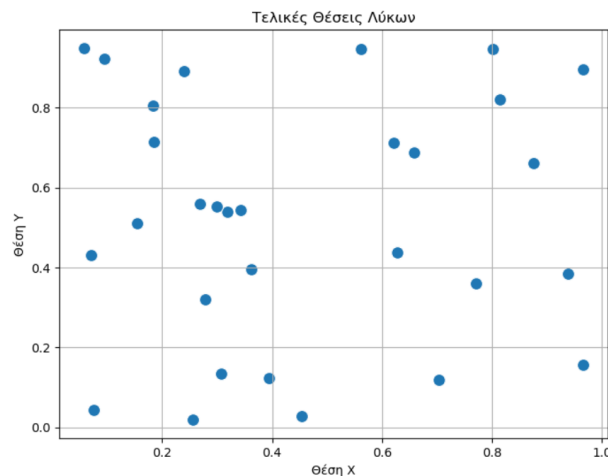
```
# Εισαγωγή των απαραίτητων βιβλιοθηκών για τη δημιουργία του
# διαγράμματος θέσεων των λύκων
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import pandas as pd

# Προσομοίωση τυχαίων θέσεων λύκων σε 100 επαναλήψεις για 30 λύκους σε
# 2D χώρο
iterations = 100
wolves_num = 30
positions = np.random.rand(iterations, wolves_num, 2) # 2D πρόβλημα,
# τυχαίες θέσεις
# Επιλογή των τελικών θέσεων των λύκων στην τελευταία επανάληψη
```

² <https://jupyter.org/>

³ <https://www.anaconda.com/>

```
final_positions = positions[-1]
# Δημιουργία DataFrame για την απεικόνιση με τη Seaborn
data = pd.DataFrame(final_positions, columns=['X', 'Y'])
# Δημιουργία του διαγράμματος θέσεων λύκων χρησιμοποιώντας Seaborn με
# στρογγυλά σημεία
plt.figure(figsize=(8, 6))
sns.scatterplot(x='X', y='Y', data=data, palette='viridis', s=100,
marker='o') # 'o' για στρογγυλά σημεία
plt.title('Τελικές Θέσεις Λύκων')
plt.xlabel('Θέση X')
plt.ylabel('Θέση Y')
plt.grid(True)
plt.show()
```



Εικόνα 14: Διάγραμμα Απεικόνισης Θέσεων Πρακτόρων Αναζήτησης στο Επίπεδο.

Επιπλέον του προαναφερθέντος διαγράμματος η βιβλιοθήκη μας δίνει και άλλες δυνατότητες όπως η κατασκευή διαγράμματος κατανομής (Error Band Plot) το οποίο θα απεικονίζει την μέση τιμή της λύσης μαζί με τις διακυμάνσεις (error bands) κατά την διάρκεια των επαναλήψεων το οποίο θα απεικονίζει την στοχαστικότητα και την διασπορά των λύσεων σε κάθε επανάληψη, επιπλέον ένα ακόμη διάγραμμα το οποίο μπορούμε να κατασκευάσουμε είναι το διάγραμμα σύγκλισης (Convergence Plot) το οποίο θα απεικονίζει της εξέλιξη των τιμών της συνάρτησης καταλληλότητας (fitness) οπτικοποιώντας έτσι την εξέλιξη των καταλληλοτήτων των λύκων και την σύγκλιση τους προς το ολικό βέλτιστο.

4.2.4 Η Βιβλιοθήκη Time

Αυτή η Βιβλιοθήκη παρέχει διάφορες λειτουργίες που σχετίζονται με τον χρόνο. Στην παρούσα εργασία θα χρησιμοποιήσουμε μία συγκεκριμένη συνάρτηση της βιβλιοθήκης την συνάρτηση `perf_counter()` που χρησιμοποιείται για ακριβείς μετρήσεις του χρόνου. Προσφέρει την πιο ακριβή μέτρηση για χρονικές διαφορές σε σχέση με την συνάρτηση

`time()` της ίδιας βιβλιοθήκης, καθώς μετράει τον χρόνο σε μικροδευτερόλεπτα (μsec) και λαμβάνει υπόψη τυχόν διακοπές στο σύστημα. [76]

Θα χρησιμοποιήσουμε την εν λόγω συνάρτηση για να μετρήσουμε τόσο τον χρόνο εκτέλεσης κάθε επανάληψης, όσο και τον συνολικό χρόνο εκτέλεσης του αλγορίθμου με όσο το δυνατόν μεγαλύτερη ακρίβεια, προκειμένου να προβούμε σε στατιστική ανάλυση και σύγκριση αποτελεσμάτων για την εξαγωγή ασφαλών συμπερασμάτων ως προς την αποδοτικότητα της υλοποίησης μας.

4.2.5 Το Πακέτο `Concurrent.futures`

Το πακέτο `Concurrent.futures` παρέχει ένα υψηλού επιπέδου API για παράλληλη επεξεργασία, η οποία επιτρέπει την εργασία με μεγάλους όγκους δεδομένων και την εκτέλεση πράξεων που εξαντλούν την μνήμη του υπολογιστή. Επίσης μπορεί να διαχειριστεί δεδομένα σε μορφή πινάκων (Arrays), πλαισίων δεδομένων (dataframes) και άλλων δομών, παρόμοια με την NumPy και την Pandas, ωστόσο σε αντίθεση με αυτές υποστηρίζει παράλληλη επεξεργασία σε πολλαπλούς πυρήνες. [77]

Το εν λόγω πακέτο μπορεί να χρησιμοποιηθεί στην παράλληλη εκτέλεση του GWO για την επιτάχυνση του αλγορίθμου σε περιπτώσεις με μεγάλο αριθμό πρακτόρων αναζήτησης (λύκων).

4.3 Λογισμικά και Εργαλεία

4.3.1 Το Οικοσύστημα Anaconda

Το Anaconda⁴ είναι ένα σύστημα διανομής Python που έχει σχεδιαστεί για την αποδοτικότερη διαχείριση και ανάπτυξη πακέτων. Το εκτεταμένο οικοσύστημά του περιλαμβάνει μια σειρά από εργαλεία, βιβλιοθήκες και πακέτα προσαρμοσμένα στην επιστήμη των δεδομένων, συμπεριλαμβανομένης της ανάλυσης δεδομένων, της μηχανικής μάθησης και του επιστημονικού υπολογισμού.

Με το Anaconda, οι προγραμματιστές και οι επιστήμονες δεδομένων μπορούν να έχουν εύκολη πρόσβαση σε μια εκτεταμένη συλλογή προεγκατεστημένων πακέτων όπως είναι οι βιβλιοθήκες NumPy, Pandas, Seaborn, Dask και άλλες. Επιπλέον, το φιλικό προς το χρήστη περιβάλλον ανάπτυξης του Anaconda καθιστά εύκολη τη δημιουργία και διαχείριση

⁴ <https://www.anaconda.com/>

απομονωμένων περιβαλλόντων, εξασφαλίζοντας την αναπαραγωγικότητα⁵ και την δυνατότητα κλιμάκωσης στα έργα δεδομένων. Το Anaconda κυκλοφορεί σε δύο εκδόσεις: την έκδοση ανοικτού κώδικα, Anaconda Individual Edition, και την εμπορική έκδοση για εταιρική χρήση με την ονομασία Anaconda Commercial Edition.

Με τις ισχυρές βιβλιοθήκες που διατίθενται στο Anaconda όπως οι SciPy, SymPy και OpenCV, οι επιστήμονες και οι ερευνητές μπορούν να επιλύουν σύνθετα μαθηματικά προβλήματα, να εκτελούν προσομοιώσεις, να επεξεργάζονται εικόνες κ.α. Επιπλέον, το NumPy, ένα θεμελιώδες πακέτο για επιστημονικούς υπολογισμούς με χρήση Python είναι προεγκατεστημένο. Ο αποτελεσματικός χειρισμός των πινάκων από αυτό το πακέτο έχει κάνει το Anaconda δημοφιλές στους επιστήμονες δεδομένων διαφόρων κλάδων. [78]

Αναφορικά με την χρήση του περιβάλλοντος Anaconda στην υλοποίηση του GWO αυτό προσφέρει απομονωμένα περιβάλλοντα για υλοποίηση αλγορίθμων με διαφορετικές εκδόσεις βιβλιοθηκών και πακέτων τα οποία δεν αλληλοεπιδρούν μεταξύ τους και δεν προκαλούν δυσχέρειες στο σύστημα, επιπροσθέτως προσφέρει ιδιαίτερη ευκολία και ευελιξία στην εγκατάσταση των βιβλιοθηκών διασφαλίζοντας τη σωστή τους εγκατάσταση μέσω του conda package manager. Το Anaconda διαθέτει ενσωματωμένο το λογισμικό Jupyter Notebook το οποίο είναι εξαιρετικά χρήσιμο στην εκτέλεση του κώδικα και την απεικόνιση των αποτελεσμάτων, πιο συγκεκριμένα προσφέρει τμηματική εκτέλεση του κώδικα για λήψη αποτελεσμάτων σε πραγματικό χρόνο, ενσωμάτωση γραφημάτων και οπτικοποίηση των αποτελεσμάτων απευθείας στο περιβάλλον του Jupyter Notebook και τέλος παραμετροποίηση του αλγορίθμου (αριθμός λύκων, μέγιστος αριθμός επαναλήψεων, διακύμανση της βέλτιστης λύσης) χωρίς την απαίτηση για διαχωρισμό των φάσεων της τροποποίησης και της επανεκτέλεσης του αλγορίθμου.

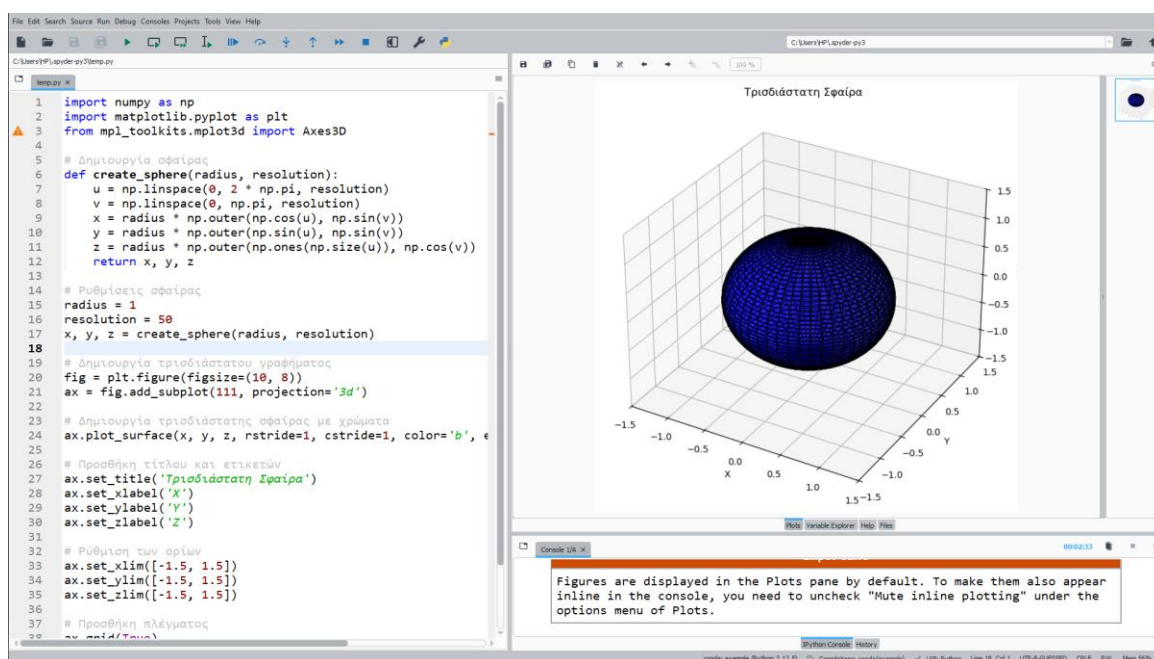
Όπως προαναφέρθηκε το Anaconda επιτρέπει την αναπαραγωγικότητα έτσι με την χρήση μίας και μόνο εντολής στην γραμμή εντολών της κονσόλας του Anaconda μπορεί να εξαχθεί ολόκληρο το περιβάλλον στο οποίο υλοποιήθηκε και εκτελείται ο αλγόριθμος και εισαγωγή του σε οποιοδήποτε άλλο σύστημα. Ομοίως το Anaconda υποστηρίζει κατανεμημένο προγραμματισμό γεγονός που δίνει την δυνατότητα για μια αποδοτικότερη υλοποίηση του αλγορίθμου με χρήση πολλών πυρήνων και την παράλληλη εκτέλεση του αλγορίθμου σε

⁵ Η αναπαραγωγικότητα αναφέρεται στην ικανότητα αναπαραγωγής ή αναδημιουργίας των αποτελεσμάτων ενός πειράματος, μιας ανάλυσης ή μιας μελέτης χρησιμοποιώντας τις ίδιες μεθόδους, δεδομένα και διαδικασίες που χρησιμοποιήθηκαν αρχικά.

αυτούς. Τέλος προσφέρει την χρήση βιβλιοθηκών για επιτάχυνση του αλγορίθμου με χρήση της GPU.

4.3.2 Το IDE Spyder

Το Spyder⁶ είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης ανοικτού κώδικα (IDE) που περιλαμβάνεται στη διανομή Anaconda και προσφέρει προηγμένες δυνατότητες επεξεργασίας, διαδραστικής δοκιμής, αποσφαλμάτωσης και ενδοσκοπήσης. Το Spyder διαλειτουργεί με δημοφιλείς βιβλιοθήκες όπως η NumPy, η SciPy, η Pandas κ.α. Επίσης επειδή είναι σχεδιασμένο για επιστήμονες δεδομένων και αναλυτές δεδομένων, διαθέτει εξαιρετικές δυνατότητες οπτικοποίησης και επεξεργασίας δεδομένων σε πραγματικό χρόνο, κάτι που είναι χρήσιμο για την ανάλυση των αποτελεσμάτων της εκτέλεσης του GWO. Συναφώς διαθέτει κατάλληλα εργαλεία όπως είναι ο variable explorer που επιτρέπει την παρακολούθηση των τιμών των μεταβλητών του αλγορίθμου σε πραγματικό χρόνο.



Εικόνα 15: Στιγμιότυπο Εκτέλεσης Κώδικα Στο IDE Spyder.

4.3.3 Το Google Colaboratory (Colab)

Το Google Collaboratory (Colab)⁷ είναι ένα εργαλείο που επιτρέπει στους χρήστες να γράφουν και να εκτελούν κώδικα Python μέσω του προγράμματος περιήγησης στο διαδίκτυο. Είναι ιδιαίτερα δημοφιλές στην μηχανική μάθηση, την ανάλυση δεδομένων και

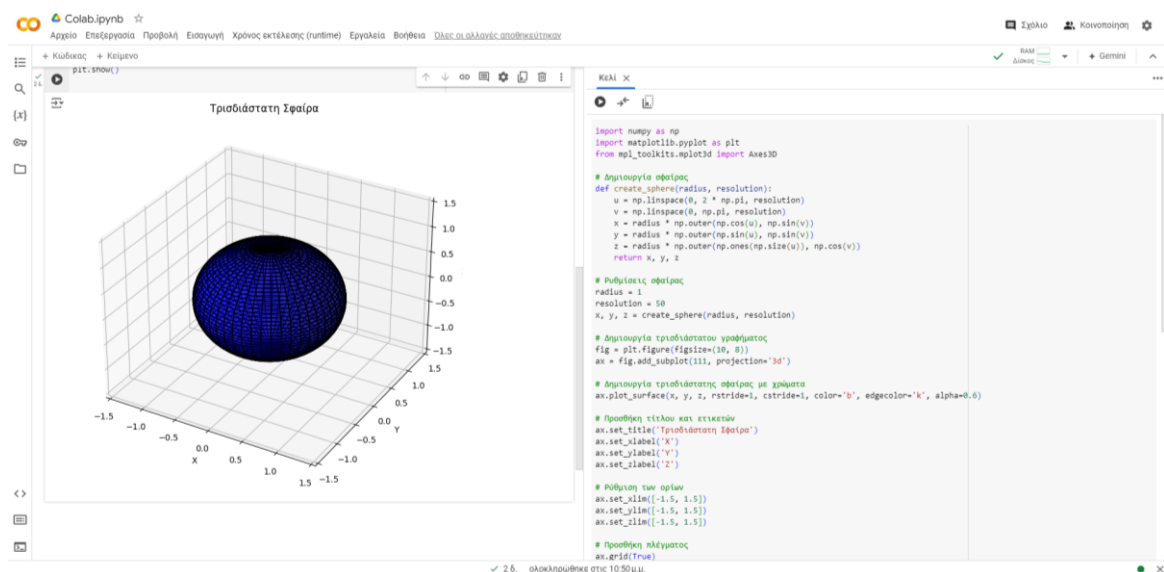
⁶ <https://www.spyder-ide.org/>

⁷ <https://colab.google/>

την εκπαίδευση, καθώς παρέχει ένα φιλικό προς τον χρήστη περιβάλλον χωρίς εγκατάσταση, με πρόσβαση σε ισχυρούς υπολογιστικούς πόρους. Το Colab συνεργάζεται με την υπηρεσία σύννεφου της Google, το Google Drive, διευκολύνοντας έτσι την κοινή πρόσβαση και τη συνεργασία σε πραγματικό χρόνο μεταξύ των χρηστών.

Το περιβάλλον Colab ενσωματώνει, όπως και το περιβάλλον Anaconda, το λογισμικό Jupyter Notebook ωστόσο σε αντίθεση με το Anaconda το Colab δεν απαιτεί την εγκατάσταση λογισμικού στον υπολογιστή μας, γεγονός που του προσδίδει το πλεονέκτημα της ευελιξίας και την χρήσης του από πολλές πλατφόρμες ανεξαρτήτως λειτουργικού συστήματος και εγκατεστημένων πακέτων. Η διαλειτουργικότητα του με το Google Drive συμβάλει επίσης στην εκτέλεση εργασιών από τον χρήστη σε οποιοδήποτε σύστημα χρησιμοποιεί χωρίς να περιορίζεται από τις υπολογιστικές ικανότητες του. [79]

Το εν λόγω εργαλείο χρησιμοποιήθηκε ιδιαίτερα στην υλοποίηση του GWO καθότι η χρήση του λογισμικού Jupyter συμπλήρωσε αυτή του Spyder σε περιπτώσεις όπου απαιτήθηκε να πραγματοποιηθούν εργασίες ανάπτυξης του κώδικα και δεν ήταν διαθέσιμο το υπολογιστικό σύστημα με εγκατεστημένο το οικοσύστημα του Anaconda.



Εικόνα 16: Στιγμιότυπο Εκτέλεσης Κώδικα Στο Google Colab.

4.4 Παρουσίαση του Αλγορίθμου

Σε αυτή την ενότητα παρουσιάζονται σε μορφή ψευδοκώδικα, αρχικά ο γενικός αλγόριθμος που υλοποιήθηκε στο πλαίσιο της παρούσης εργασίας και ακολούθως παρουσιάζονται αναλυτικότερα οι βασικές συναρτήσεις του. Συναφώς παρατίθεται αρχικά και ένα

διάγραμμα ροής προγράμματος που σκοπό έχει την συμβολή στην κατανόηση του τρόπου λειτουργίας του αλγορίθμου από τον αναγνώστη.

Επισημαίνεται ότι ο υπολογισμός των καταλληλοτήτων των λύκων αποτελεί μία ιδιαίτερα απαιτητική εργασία σε όρους υπολογιστικού κόστους, ειδικά αν ο πληθυσμός είναι μεγάλος και οι συναρτήσεις αξιολόγησης είναι πολύπλοκες καθότι οι υπολογισμοί αυτοί επαναλαμβάνονται σε κάθε επανάληψη του αλγορίθμου. Για να αντιμετωπίσουμε αυτό το μειονέκτημα της σειριακής εκτέλεσης του υπολογισμού αξιοποιήθηκε η δυνατότητα παράλληλης επεξεργασίας που προσφέρει η γλώσσα με χρήση του πακέτου `concurrent.futures` όπως αναφέρεται στην υποενότητα 4.2.5 η οποία αναμένεται να οδηγήσει σε αισθητή μείωση του χρόνου εκτέλεσης καθότι είναι γνωστό πως η παράλληλη επεξεργασία επιταχύνει τέτοιου είδους υπολογισμών σε πολυπύρρηνα συστήματα.

Χρησιμοποιήθηκε η κλάση `ThreadPoolExecutor` για το σκοπό αυτό με την δημιουργία ενός `pool` από νήματα τα οποία ανέλαβαν και τον υπολογισμό της καταλληλότητας για ισάριθμους λύκους παράλληλα κάθε στιγμή. Ο υπολογισμός γίνεται μέσα στην συνάρτηση `find_best_three_wolves()` με χρήση της μεθόδου `executor.map()` όπως φαίνεται στο παράρτημα Α στο αρχείο `gwo.py`.

```
PROCEDURE GWO(wolves_number, max_iteration_num,  
min_fitness_standard_deviation, benchmark_function, plot_graph_enabled)  
  # 1. Αρχικοποίηση των θέσεων των λύκων τυχαία.  
  INITIALIZE wolves_positions_array randomly  
  
  # 2. Αρχικοποίηση του μετρητή επαναλήψεων.  
  INITIALIZE iteration_counter = 0  
  
  # 3. Εκτέλεση του αλγορίθμου έως ότου εξαντληθούν οι επαναλήψεις ή  
  ικανοποιηθεί η συνθήκη σύγκλισης.  
  WHILE iteration_counter < max_iteration_num AND  
  fitness_standard_deviation > min_fitness_standard_deviation  
    # 4. Εύρεση των 3 καλύτερων λύκων  
    best_wolves = find_best_three_wolves(wolves_positions_array)  
  
    # 5. Ενημέρωση των θέσεων όλων των λύκων  
    FOR EACH wolf IN wolves_positions_array  
      # Υπολογισμός νέας θέσης για κάθε λύκο  
      UPDATE wolf position using best wolves information  
    END FOR  
  
    # 6. Μείωση του παραμέτρου εξερεύνησης (a_factor) για καλύτερη  
    σύγκλιση.  
    DECREASE exploration_factor (a_factor)
```

```
# 7. Εύρεση των 3 καλύτερων λύκων μετά την ενημέρωση των θέσεων.
best_wolves = find_best_three_wolves(wolves_positions_array)

# 8. Αύξηση του μετρητή επαναλήψεων.
iteration_counter += 1
END WHILE

# 9. Υπολογισμός τελικών αποτελεσμάτων του αλγορίθμου.
results = CALCULATE final_results(best_wolves,
wolves_positions_array)

# 10. Επιστροφή αποτελεσμάτων
RETURN results
END PROCEDURE

PROCEDURE find_best_three_wolves(wolves_positions_array)
# 1. Αρχικοποίηση των τριών καλύτερων λύκων (α, β, δ).
INITIALIZE best_wolves = [alpha, beta, delta]

# 2. Υπολογισμός καλύτερων λύκων με βάση τις καταλληλότητες.
FOR EACH wolf IN wolves_positions_array
# Ενημέρωση των καλύτερων λύκων με βάση την καταλληλότητα του
κάθε λύκου
UPDATE best_wolves based on wolf fitness
END FOR

# 3. Επιστροφή των 3 καλύτερων λύκων.
RETURN best_wolves
END PROCEDURE

PROCEDURE calculate_new_position(best_wolves, wolf_position)
# 1. Υπολογισμός της νέας θέσης του λύκου με βάση τις θέσεις των
τριών καλύτερων λύκων.
CALCULATE new position using best wolves and random factors

# 2. Επιστροφή της νέας θέσης.
RETURN new position
END PROCEDURE

PROCEDURE calculate_fitness(wolf_position)
# 1. Υπολογισμός της καταλληλότητας για τη δεδομένη θέση του λύκου.
CALCULATE fitness for the given wolf position

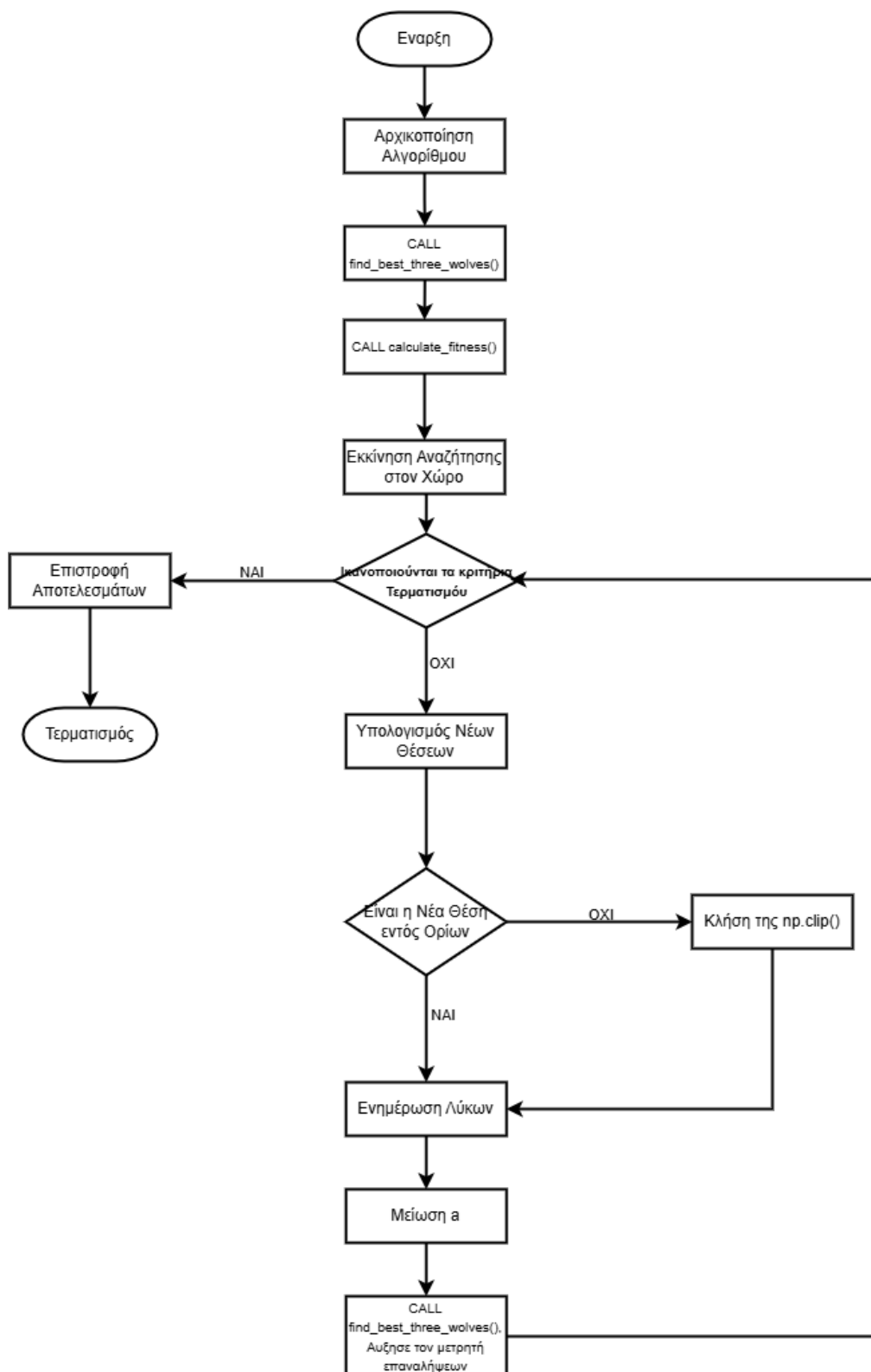
# 2. Επιστροφή της καταλληλότητας
RETURN fitness value
END PROCEDURE

PROCEDURE calculate_execution_time()
# 1. Υπολογισμός του χρόνου εκτέλεσης του αλγορίθμου
CALCULATE execution time

# 2. Επιστροφή του χρόνου εκτέλεσης
RETURN time
END PROCEDURE
```

Ψευδοκώδικας 4: Εποπτικός Ψευδοκώδικας του Αλγορίθμου

Επιπροσθέτως παρατίθεται και ένα Διάγραμμα Ροής που παρέχει μία εποπτική αναπαράσταση της εκτέλεση του αλγορίθμου



Εικόνα 17: Διάγραμμα Ροής GWO

Παρακάτω παρουσιάζονται σε μορφή ψευδοκώδικα οι βασικές συναρτήσεις του αλγορίθμου. Αρχικά παρουσιάζεται η βασική συνάρτηση GWO() στην οποία γίνεται η αρχικοποίηση των λύκων στον χώρο αναζήτησης με όσο το δυνατόν πιο τυχαίο τρόπο για την επίτευξη της βέλτιστης ποικιλομορφίας των λύσεων και της βέλτιστης εξερεύνησης του χώρου προς αποφυγή εγκλωβισμού σε τοπικό ελάχιστο. Ακολούθως πραγματοποιείται η αρχικοποίηση βασικών παραμέτρων του GWO όπως είναι ο παράγοντας a που επηρεάζει την λειτουργία του αλγορίθμου και την εναλλαγή μεταξύ εξερεύνησης και εκμετάλλευσης.

Εν συνεχεία υπολογίζονται για πρώτη φορά οι τρεις καλύτερες λύσεις από τις τυχαία τοποθετημένες στον χώρο αναζήτησης με κλήση της συνάρτησης `find_best_three_wolves()` η οποία και υπολογίζει τις καταλληλότητες όλων των πρακτόρων αναζήτησης και επιστρέφει τους τρεις με τις μικρότερες καταλληλότητες. Υπολογίζει τις νέες θέσεις όλων των υπολοίπων πρακτόρων αναζήτησης με βάση τους τρεις καλύτερους πράκτορες καλώντας την συνάρτηση `calculate_new_position()`, εν συνεχεία γίνεται έλεγχος και τροποποίηση των θέσεων που είναι εκτός ορίων χώρου αναζήτησης με την συνάρτηση `np.clip()`.

Στο επόμενο βήμα μειώνεται ο παράγοντας a προκειμένου να μεταβεί ο αλγόριθμος σταδιακά από την εξερεύνηση του χώρου στην εκμετάλλευση της εντοπισθείσας υποψήφιας βέλτιστης λύσης και επαναυπολογίζονται οι καλύτεροι πράκτορες της τρέχουσας επανάληψης. Όταν ικανοποιηθούν τα κριτήρια τερματισμού ο αλγόριθμος θα τερματίσει. Τέλος υπολογίζονται κάποιες μετρικές της απόδοσης του αλγορίθμου όπως είναι η απόκλιση της καταλληλότητας της λύσης που υπολόγισε ο αλγόριθμος και της πραγματικά βέλτιστης λύσης. Συγκεκριμένα χρησιμοποιείται η ευκλείδεια απόσταση για τον υπολογισμό αυτής της τιμής. Η συνάρτηση ολοκληρώνεται με την εκτέλεση κώδικα για την αποθήκευση των δεδομένων του αλγορίθμου σε ένα λεξικό και την κλήση της συνάρτησης εκτύπωσης αυτών των δεδομένων σε πινακοειδή μορφή στην κονσόλα.

PROCEDURE GWO(wolves_number, iteration_number, min_fitness_variance):

```
# 1. Καταγραφή του χρόνου έναρξης της εκτέλεσης του αλγορίθμου
start_time = CALL current_time()

# 2. Δημιουργία πίνακα θέσεων των λύκων
wolves_positions = CALL create_random_positions(wolves_number,
variables_number, lower_bound, upper_bound)
```

```
# 3. Αρχικοποίηση παραμέτρων για την ισορροπία
εξερεύνησης/εκμετάλλευσης
a_factor = 2
iteration_count = 0 # Αρχικός μετρητής επαναλήψεων

# 4. Εύρεση των καλύτερων λύκων (Χα, Χβ, Χγ) και της διακύμανσης
fitness
best_wolves, fitness_variance, wolves_fitness = CALL
find_best_three_wolves(wolves_positions, iteration_count)

# 5. Εκτέλεση της αναζήτησης έως ότου ικανοποιηθούν οι συνθήκες
τερματισμού
WHILE iteration_count < max_iteration_num and fitness_variance >
min_fitness_variance:

    # 5.1 Ενημέρωση θέσεων για κάθε λύκο στον πληθυσμό
    FOR EACH wolf in population:
        FOR EACH dimension:
            # Υπολογισμός νέας θέσης για τον λύκο με βάση τις
            καλύτερες θέσεις (Χα, Χβ, Χγ)
            new_position = CALL calculate_new_position(best_wolves,
            each_wolf[dimension], a_factor, dimension)

            # Έλεγχος αν η νέα θέση είναι εντός των επιτρεπτών ορίων
            each_wolf[dimension] = CALL
            check_new_position(new_position)
        END_FOR
    END_FOR

    # 5.2 Ενημέρωση της παραμέτρου a σε κάθε επανάληψη
    a_factor = 2 - (2 * iteration_count / max_iteration_num)

    # 5.3 Ενημέρωση των καλύτερων λύκων και υπολογισμός της
    διακύμανσης fitness
    best_wolves, fitness_variance, wolves_fitness = CALL
    find_best_three_wolves(wolves_positions, iteration_count)

    # 5.4 Αύξηση του μετρητή επαναλήψεων
    INCREMENT iteration_count

END_WHILE

# 6. Καταγραφή του χρόνου ολοκλήρωσης της εκτέλεσης
end_time = CALL current_time()
gwo_time = end_time - start_time

# 7. Υπολογισμός της απόκλισης από το ολικό ελάχιστο
positional_deviation = CALL
calculate_euclidean_distance(best_wolves_positions, global_minimum)

# 8. Υπολογισμός της τιμής του fitness στο ολικό ελάχιστο
discovered_minimum_value = CALL
benchmark_function.calculate_fitness(global_minimum)

# 9. Δημιουργία λεξικού αποτελεσμάτων
results = {
    "global_minimum": global_minimum,
    "discovered_minimum_value": benchmark_function.actual_minimum,
    "best_wolves": best_wolves,
    "mean_fitness": calculate_mean(wolves_fitness),
    "min_fitness": calculate_min(wolves_fitness),
```

```
"positional_deviation": positional_deviation,  
"discovered_minimum_value": discovered_minimum_value,  
"iterations": iteration_count,  
"execution_time": gwo_time,  
}
```

```
# 10. Εκτύπωση των αποτελεσμάτων  
CALL print_results(results)
```

END PROCEDURE

Ψευδοκώδικας 5: Βασική Συνάρτηση GWO() του Αλγορίθμου

Ακολουθώς παρατίθεται σε μορφή ψευδοκώδικα η συνάρτηση στην οποία πραγματοποιείται ο υπολογισμός των τριών καλύτερων πρακτόρων αναζήτησης που βρίσκονται στον χώρο των λύσεων. Για να υπολογιστούν οι ηγέτες λύκοι πραγματοποιείται υπολογισμός της καταλληλότητας για τον κάθε πράκτορα αναζήτησης γεγονός το οποίο αναλόγως του προβλήματος και του πλήθους των πιθανών λύσεων μπορεί να είναι ιδιαίτερα χρονοβόρο. Για να αντιμετωπισθούν αυτές οι δυσκολίες ακολουθήθηκε η πρακτική του παράλληλου προγραμματισμού με την χρήση της κατάλληλης για τον σκοπό αυτό βιβλιοθήκης. Τα βήματα που εκτελεί η εν λόγω συνάρτηση επισημαίνονται με κατάλληλο σχόλιο εντός του ψευδοκώδικα παρακάτω. Επισημαίνεται ότι για τον υπολογισμό της καταλληλότητας με χρήση κάποιας αντικειμενικής συνάρτησης χρησιμοποιείται ένα υποσύνολο από κάποιες γνωστές συναρτήσεις δοκιμής (Benchmark Functions) οι οποίες περιγράφονται στο επόμενο κεφάλαιο.

```
PROCEDURE find_best_three_wolves(wolves_positions_array,  
iteration_counter)  
# 1. Αρχικοποίηση του λεξικού με τους καλύτερους λύκους.  
INITIALIZE best_wolves_dictionary:  
0 : [zero vector, +∞]  
1 : [zero vector, +∞]  
2 : [zero vector, +∞]  
  
# 2. Υπολογισμός καταλληλότητων με παράλληλη εκτέλεση.  
CREATE thread_pool_executor  
FOR EACH wolf IN population  
APPLY calculate_fitness()  
STORE results IN wolves_fitness_list  
END_FOR  
CLOSE thread_pool_executor  
  
#3. Μετατροπή της λίστας με τις καταλληλότητες των Πρακτόρων σε  
#Πίνακα Numpy  
CONVERT wolves_fitness_list TO numpy array  
wolves_fitness_numpy_array  
  
#4. Υπολογισμός των τριών καλύτερων πρακτόρων αναζήτησης  
FOR EACH fitness IN wolves_fitness_numpy_array
```

```
IF current_wolves_fitness < alpha_wolf_fitness THEN
    delta_wolf = beta_wolf
    beta_wolf = alpha_wolf
    alpha_wolf = current_wolf
ELSE IF current_wolves_fitness < beta_wolf_fitness THEN
    Delta_wolf = beta_wolf
    Beta_wolf = current_wolf
ELSE IF current_wolf_fitness < delta_wolf_fitness THEN
    Delta_wolf = current_wolf
END IF
END FOR

#5. Σχεδιασμός Γραφήματος υπο προϋποθέσεις
IF variables_number = 2 AND iteration_counter % 2 = 0 THEN
    CALL plot_graph(wolves_positions_array,
wolves_fitness_numpy_array, iteration_counter)
END IF

#6. Υπολογισμός της Διακύμανσης της Καταλληλότητας

fitness_variance = CALCULATE VARIANCE OF wolves_fitness_numpy_array

#7. Επιστροφή του λεξικού με τους 3 καλύτερους λύκους, της Διακύμανσης
της Καταλληλότητας καθώς και ολόκληρου του πίνακα των καταλληλοτήτων
στην τρέχουσα επανάληψη.
RETURN best_wolves_dictionary, fitness_variance,
wolves_fitness_numpy_array

END PROCEDURE
```

Ψευδοκώδικας 6: Συνάρτηση Υπολογισμού των Καλύτερων Λύκων

Παρακάτω παρατίθεται σε μορφή ψευδοκώδικα η συνάρτηση `calculate_new_position()` η οποία λαμβάνει ως παραμέτρους το λεξικό με τους τρεις καλύτερους μέχρι στιγμής λύκους, την τρέχουσα θέση του λύκου που θέλουμε να υπολογίσουμε την νέα θέση, του παράγοντα a και έναν ακέραιο που αντιπροσωπεύει την διάσταση στην οποία θέλουμε να υπολογίσουμε το νέα συντεταγμένη για παράδειγμα ένα έχουμε δύο διαστάσεις στο πρόβλημα μας τότε η εν λόγω μεταβλητή θα πάρει τιμές 1 και 2 που θα αντιστοιχούν στους άξονες x και y . Ακολούθως στο σώμα της συνάρτησης αρχικοποιούνται οι παράμετροι $r1$ και $r2$, υπολογίζονται οι παράμετροι A , C και D και ακολούθως για την τρέχουσα διάσταση υπολογίζονται οι τρεις νέες συντεταγμένες του λύκου για κάθε έναν ηγέτη λύκο, ενώ ο μέσος όρος αυτών είναι η νέα θέση του λύκου για την δεδομένη διάσταση. Τέλος επιστρέφεται στην βασική συνάρτηση `GWO()` η τιμή της νέας θέσης.

```
PROCEDURE calculate_new_position(best_wolves_dictionary, wolf_position,
a, every_dimension)
```

```
# 1. Δημιουργία δύο τυχαίων αριθμών r1 και r2 με ομοιόμορφη κατανομή
r1, r2 = CALL generate_uniform_random_numbers(2)
```

2. Υπολογισμός της παραμέτρου A που επηρεάζει τη διαδικασία αναζήτησης και σύγκλισης

```
A = 2 * a * r1 - a
```

3. Υπολογισμός της παραμέτρου C που καθορίζει την επίδραση των καλύτερων λύκων

```
C = 2 * r2
```

4. Υπολογισμός της απόστασης D από τους τρεις καλύτερους λύκους

```
D_alpha = ABS(C * alpha_wolf_position - wolf_position)
```

```
D_beta = ABS(C * beta_wolf_position - wolf_position)
```

```
D_delta = ABS(C * delta_wolf_position - wolf_position)
```

5. Υπολογισμός των νέων θέσεων με βάση τους τρεις καλύτερους λύκους

```
wolf_position1 = alpha_wolf_position - A * D_alpha
```

```
wolf_position2 = beta_wolf_position - A * D_beta
```

```
wolf_position3 = delta_wolf_position - A * D_delta
```

6. Υπολογισμός της νέας θέσης ως ο μέσος όρος των θέσεων από τους τρεις καλύτερους λύκους

```
wolf's_new_position = (wolf_position1 + wolf_position2 +  
wolf_position3) / 3
```

7. Επιστροφή της νέας θέσης για τη συγκεκριμένη διάσταση

```
RETURN wolf's_new_position
```

END PROCEDURE

Ψευδοκώδικας 7: Συνάρτηση Υπολογισμού Νέας Θέσης Λύκου

Στο τέλος της εργασίας στο Παράρτημα «Α» παρατίθεται ο κώδικας του υλοποιημένου GWO σε Python, επαρκώς σχολιασμένος με επεξηγηματικά σχόλια και με χρήση docstrings με στυλ Google [80] σε όλες τις βασικές μεθόδους για την πληρέστερη παρουσίαση και κατανόηση της λειτουργίας του αλγορίθμου από τον Αναγνώστη. Συναφώς αναπτύχθηκε και ένα απλό γραφικό περιβάλλον (GUI) για βελτίωση της εμπειρίας του χρήστη κατά την παραμετροποίηση του αλγορίθμου αλλά και κατά την φάση της εκτέλεσης του. Μια σύντομη παρουσίαση της GUI πραγματοποιείται στο παράρτημα «Γ».

Στο 5^ο κεφάλαιο που ακολουθεί αρχικά γίνεται εισαγωγή στις συναρτήσεις αξιολόγησης αλγορίθμων βελτιστοποίησης και ακολούθως παρουσιάζονται οι συναρτήσεις οι οποίες επιλέχθηκαν για την αξιολόγηση του GWO. Τα τεχνικά χαρακτηριστικά του συστήματος στο οποίο διεξήχθησαν οι δοκιμές παρατίθενται στο Παράρτημα «Β», ενώ τα συμπεράσματα που προέκυψαν παρουσιάζονται στο 6^ο κεφάλαιο.

5. Αξιολόγηση του Αλγορίθμου

Σύμφωνα με τους Sharma και Raju [81] οι συναρτήσεις δεικτών αναφοράς (benchmarks functions) διαδραματίζουν σημαντικό ρόλο στον τομέα των αλγορίθμων βελτιστοποίησης. Οι δείκτες αυτοί στην υπάρχουσα βιβλιογραφία αναφέρονται συχνά και ως συναρτήσεις δοκιμής ή σουίτες δοκιμών (test suites) χρησιμοποιούνται για την αξιολόγηση και την επικύρωση της απόδοσης των αλγορίθμων βελτιστοποίησης. Αυτές οι συναρτήσεις δοκιμής είναι ουσιαστικά τεχνητά προβλήματα βελτιστοποίησης με γνωστές λύσεις και χώρους αναζήτησης που έχουν σχεδιαστεί με σκοπό την αξιολόγηση της απόδοσης των αλγορίθμων.

Σύμφωνα με την υπάρχουσα βιβλιογραφία σύνολα συναρτήσεων δοκιμής χρησιμοποιούνται για την επικύρωση και τη σύγκριση διαφόρων αλγορίθμων βελτιστοποίησης και εφόσον ένας αλγόριθμος βελτιστοποίησης επιδεικνύει υψηλή απόδοση σε αυτά τα σύνολα είναι πιθανό να επιδείξει παρόμοια συμπεριφορά και σε προβλήματα του πραγματικού κόσμου.

5.1 Εισαγωγή στις Συναρτήσεις Αξιολόγησης

Ένας μεταερευνητικός αλγόριθμος όπως ο GWO μπορεί να αντιμετωπίσει αρκετές δυσκολίες κατά την προσέγγιση στο ολικό βέλτιστο ενός προβλήματος. Λόγω της άγνωστης μορφής του χώρου αναζήτησης των προβλημάτων του πραγματικού κόσμου είναι αδύνατο να παρατηρηθούν αμέσως τα χαρακτηριστικά των χώρων αναζήτησης και κατά συνέπεια η συμπεριφορά των μεταερευνητικών αλγορίθμων. Ως εκ τούτου, οι ερευνητές χρησιμοποιούν συνήθως συναρτήσεις δοκιμής κατά την ανάπτυξη, βελτίωση ή ανάλυση ενός αλγορίθμου σε αυτόν τον τομέα.

Μερικές από τις σημαντικότερες δυσκολίες που σχετίζονται με την επίλυση πραγματικών προβλημάτων είναι η αργή σύγκλιση, ο μεγάλος αριθμός τοπικών βέλτιστων, ο μεγάλος αριθμός μεταβλητών, η εξάρτηση μεταξύ μεταβλητών, οι διάφοροι περιορισμοί, οι παραπλανητικοί χώροι αναζήτησης, οι επίπεδοι χώροι αναζήτησης και οι αβεβαιότητες. Η βιβλιογραφία δείχνει ότι οι ερευνητές ανέπτυξαν διάφορα προβλήματα δοκιμής για την προσομοίωση αυτών των χαρακτηριστικών των πραγματικών χώρων αναζήτησης.

Για τη συγκριτική αξιολόγηση του ρυθμού σύγκλισης ενός αλγορίθμου, έχουν σχεδιαστεί και χρησιμοποιηθεί μονότροπες συναρτήσεις Αξιολόγησης (unimodal benchmark

functions). Μια μονότροπη συνάρτηση δοκιμής έχει ένα μόνο ολικό βέλτιστο και δεν υπάρχουν άλλα τοπικά βέλτιστα στο χώρο αναζήτησης. Επιπλέον, ολόκληρος ο χώρος αναζήτησης ευνοεί αυτό το μοναδικό βέλτιστο. Μερικές από τις πιο δημοφιλείς μονότροπες συναρτήσεις είναι οι συναρτήσεις Sphere, Easom, Beale's, Matyas και McCormik. Οι μονότροπες συναρτήσεις δοκιμής συγκρίνουν τόσο την ταχύτητα σύγκλισης όσο και την εκμετάλλευση (exploitation) του εκάστοτε αλγορίθμου.

Ο μεγάλος αριθμός τοπικών βέλτιστων προσομοιώνεται με την χρήση των πολυτροπικών και σύνθετων συναρτήσεις δοκιμής. Μια πολυτροπική συνάρτηση δοκιμής έχει πολλαπλά βέλτιστα εκ των οποίων το ένα είναι το ολικό και τα υπόλοιπα είναι τοπικά και διαθέτουν συνήθως εκθετικά αυξανόμενο αριθμό τοπικών βέλτιστων αναλόγως του αριθμού των μεταβλητών. Ορισμένες από τις πιο δημοφιλείς πολύτροπες συναρτήσεις είναι η Ackley, η Schaffer, η Griewank, η Rastrigin κ.α. Οι σύνθετες συναρτήσεις δοκιμής ανήκουν επίσης στην οικογένεια των πολυτροπικών συναρτήσεων δοκιμής, οι οποίες προκύπτουν από τον συνδυασμό, την μετατόπιση και την περιστροφή άλλων μονότροπων και πολύτροπων συναρτήσεων.

Η δυσκολία ενός προβλήματος αυξάνεται ανάλογα με το πλήθος των μεταβλητών. Ο μεγάλος αριθμός μεταβλητών αποτελεί συνήθως χαρακτηριστικό των δοκιμαστικών συναρτήσεων. Επομένως, ο επιθυμητός αριθμός μεταβλητών μπορεί εύκολα να προκύψει από το άθροισμα ή τον πολλαπλασιασμό των παραμέτρων. Μια ακόμη δυσχέρεια είναι ότι οι μεταβλητές μπορεί να έχουν εξαρτήσεις.

Οι παραπλανητικοί χώροι αναζήτησης και οι επίπεδοι χώροι αναζήτησης είναι δύο ειδικά χαρακτηριστικά που μπορεί να εμφανιστούν κατά την επίλυση ενός πραγματικού προβλήματος. Στην πρώτη περίπτωση, ολόκληρος ο χώρος αναζήτησης παρέχει παραπλανητικές πληροφορίες σχετικά με τη θέση του ολικού βέλτιστου. Στη δεύτερη περίπτωση, ο χώρος αναζήτησης παρέχει πολύ λίγες και μερικές φορές καθόλου πληροφορίες σχετικά με την πιθανή θέση του ολικού βέλτιστου. Αμφότερες αυτές οι περιπτώσεις προσομοιώνονται από πολύ δύσκολες συναρτήσεις δοκιμής.

Τέλος, οι αβεβαιότητες είναι πολύ κρίσιμες ανεπιθύμητες εισόδους για τα προβλήματα βελτιστοποίησης. Οι αβεβαιότητες μπορούν να εμφανιστούν στις εισόδους, στις εξόδους, στους περιορισμούς και τις συνθήκες λειτουργίας των προβλημάτων βελτιστοποίησης. Η διαδικασία εξέτασης οποιασδήποτε από αυτές τις αβεβαιότητες κατά τη βελτιστοποίηση

ονομάζεται ισχυρή βελτιστοποίηση (robust optimization). Η ισχυρή βελτιστοποίηση μας επιτρέπει να διασφαλίσουμε ότι η τιμή της αντικειμενικής συνάρτησης της ολικά βέλτιστης λύσης θα παραμείνει σταθερή σε περίπτωση ύπαρξης αβεβαιοτήτων. [82]

5.2 Συναρτήσεις Αξιολόγησης για τον GWO

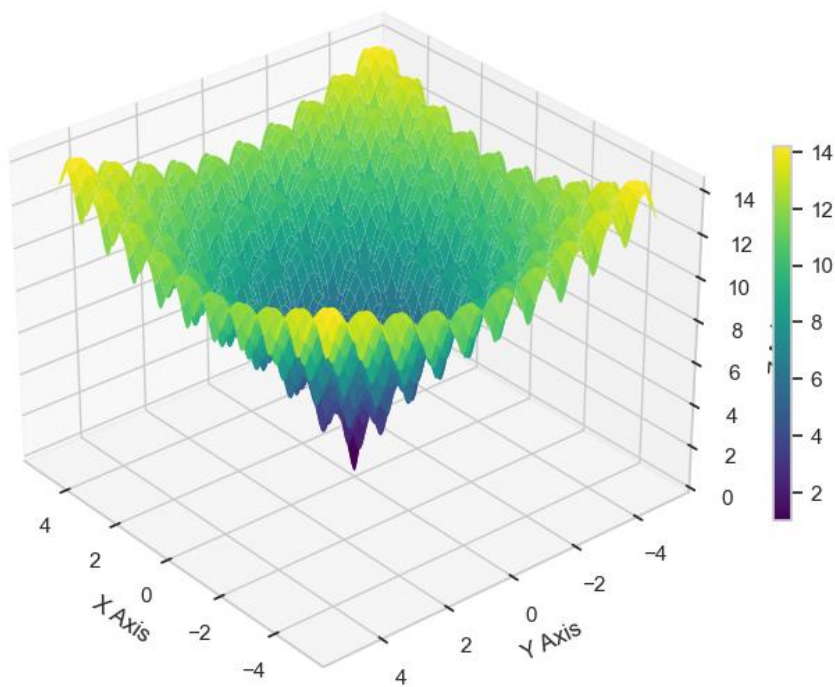
Στην παρούσα ενότητα θα αναφερθούμε στις συναρτήσεις αξιολόγησης που χρησιμοποιήθηκαν για τον έλεγχο της απόδοσης του υλοποιημένου αλγορίθμου GWO, επίσης θα χρησιμοποιηθούν για την μέτρηση της απόδοσης και άλλων επιλεγμένων μεταερευνητικών αλγορίθμων για σκοπούς σύγκρισης και εξαγωγής συμπερασμάτων σχετικά με την αποδοτικότητα η μη του GWO σε σχέση με άλλους μεταερευνητικούς.

5.2.1 Συνάρτηση Ackley (Ackley Function)

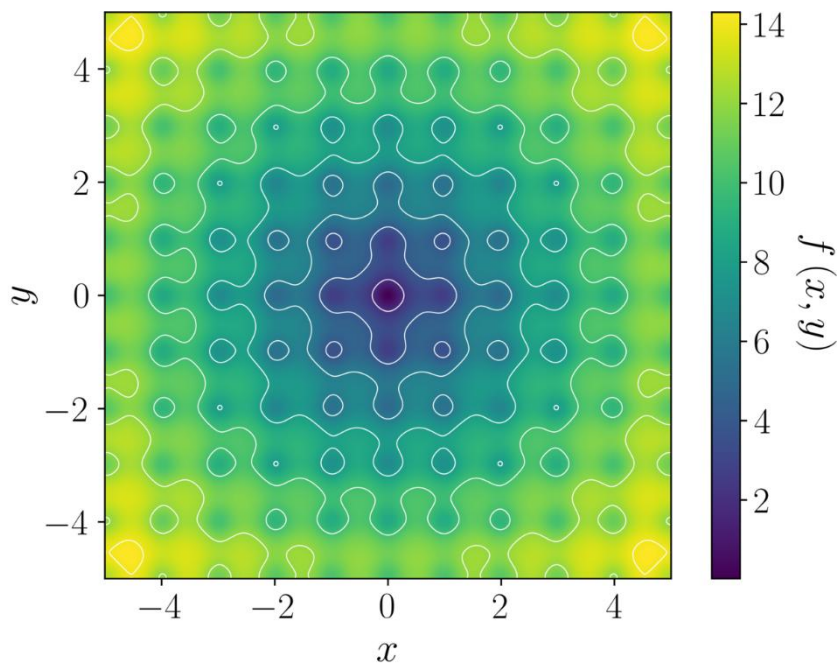
Η συνάρτηση Ackley χρησιμοποιείται ευρέως για την δοκιμή αλγορίθμων βελτιστοποίησης. Στην δισδιάστατη μορφή της όπως φαίνεται και στο σχήμα παρακάτω χαρακτηρίζεται από μία σχεδόν επίπεδη επιφάνεια στην εξωτερική περιοχή και από ένα μία απότομη και βαθιά καταβόθρα στο κέντρο η οποία αποτελεί και το ολικό βέλτιστο. Η συνάρτηση ενέχει το ρίσκο εγκλωβισμού σε ένα από τα πολλά τοπικά ακρότατα, όπως έχουμε αναφέρει ο GWO είναι επιρρεπής στην στασιμότητα και στον εγκλωβισμό σε τοπικά ακρότατα ως εκ τούτου αποτελεί καλή επιλογή για τον έλεγχο της υλοποίησης του GWO. Επιπροσθέτως η εν λόγω συνάρτηση μπορεί να οριστεί και σε χώρο πολλών διαστάσεων γεγονός που εξυπηρετεί καθότι ο GWO είναι εξ αρχής σχεδιασμός να λειτουργεί και σε πολυδιάστατους χώρους αναζήτησης. [83]

Το πεδίο ορισμού της συνάρτησης είναι συνήθως το διάστημα $[-32.768, 32.768]$ ωστόσο μπορεί και να περιοριστεί και σε ένα μικρότερο διάστημα προκειμένου να μειωθεί το κόστος εκτέλεσης του αλγορίθμου. Στο πλαίσιο αυτής της εργασίας θα χρησιμοποιήσουμε την δισδιάστατη μορφή της συνάρτησης με δύο μεταβλητές και στο διάστημα $[-32.768, 32.768]$. Ενώ θα χρησιμοποιήσουμε και τις προτεινόμενες από την βιβλιογραφία τιμές για τις παραμέτρους a, b και c . Έτσι έχουμε $a = 20, b = 0.2, c = 2\pi$. Το ολικό ελάχιστο ανεξαρτήτως διαστάσεων και άρα για $x^* = 0, 0, 0, \dots, 0$ είναι το $f(x^*) = 0$ έτσι και στην δισδιάστατη εκδοχή έχουμε ότι $f(0,0) = 0$. Η συνάρτηση με τις προκαθορισμένες τιμές των παραμέτρων a, b, c είναι η εξής:

$$F_{10} = f(x, y) = -20 \cdot e^{-0.2 \cdot \sqrt{0.5 \cdot (x^2 + y^2)}} - e^{0.5 \cdot (\cos 2\pi x + \cos 2\pi y)} + e + 20 \quad (4.1)$$



Εικόνα 18: Γράφημα Συνάρτησης Ackley.



Εικόνα 19: Γράφημα Κάτοψης Συνάρτησης Ackley

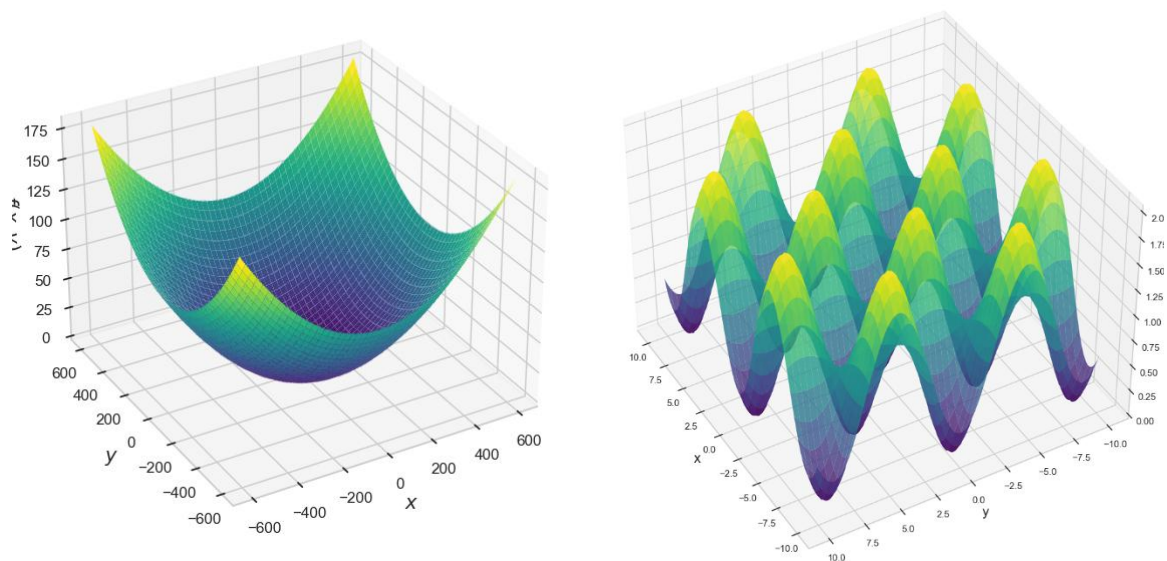
5.2.2 Συνάρτηση Griewank (Griewank Function)

Η συνάρτηση Griewank διαθέτει πολλά τοπικά ελάχιστα διασκορπισμένα στον χώρο με μεγάλη πολυπλοκότητα αναφορικά με την συχνότητα εμφάνισης τους όπως φαίνεται και στην κάτωθι δεξιά εικόνα η οποία είναι μία μεγέθυνση της αριστερής. Το γεγονός αυτό προκαλεί δυσκολία στους αλγορίθμους για την εύρεση του ολικού βέλτιστου. Το πεδίο ορισμού είναι το διάστημα $[-600,600]$. Ομοίως με την συνάρτηση Ackley το ολικό ελάχιστο είναι το $f(x, y) = 0$ για $x = y = 0$. [83]

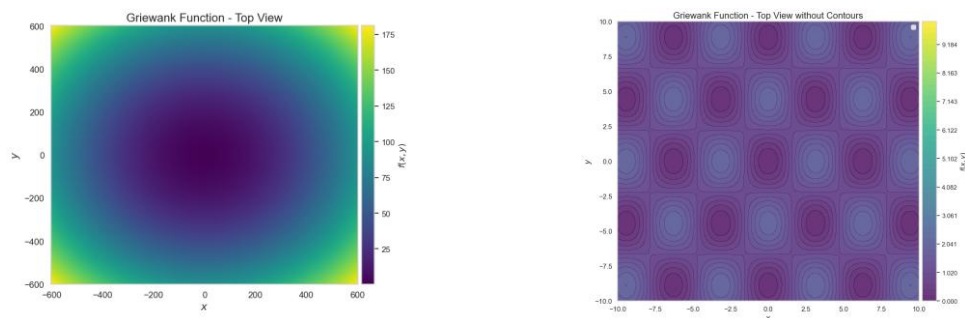
Ο τύπος της συνάρτησης είναι ο εξής:

$$F_{11} = f(x, y) = \sum_{i=1}^d \frac{x_i^2}{4000} - \prod_{i=1}^d \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1 \quad (4.2)$$

Τα διαγράμματα της συνάρτησης φαίνονται ακολούθως:



Εικόνα 20: Γράφημα Συνάρτησης Griewank



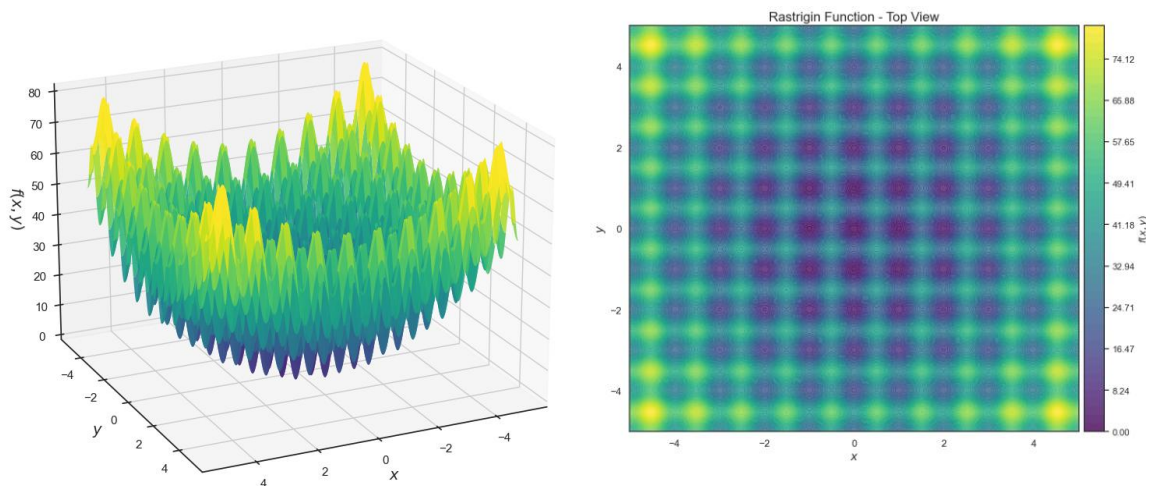
Εικόνα 21: Γραφήματα Σκιάς Συνάρτησης Griewank

5.2.3 Συνάρτηση Rastrigin (Rastrigin Function)

Η συνάρτηση Rastrigin αρχικά προτάθηκε σε δισδιάστατη μορφή από τον Rastrigin και ακολούθως γενικεύθηκε από μεταγενέστερους ερευνητές. Η εν λόγω συνάρτηση είναι μία πολυτροπική συνάρτηση (multimodal function) δηλαδή διαθέτει πολλά τοπικά ελάχιστα γεγονός που σε συνδυασμό με τον μεγάλο χώρο αναζήτησης καθιστά την εύρεση του ολικού βέλτιστου μία δύσκολη υπόθεση. Το πεδίο ορισμού της συνάρτησης είναι το $[-5.12, 5.12]$ και το ολικό ελάχιστο ανεξαρτήτως διαστάσεων είναι το $f(0,0,0, \dots, 0) = 0$. [83]

$$f(x) = 10 \cdot d + \sum_{i=0}^d [x_i^2 - 10 \cdot \cos(2\pi x_i)] \quad (4.3)$$

όπου d είναι η διάσταση.



Εικόνα 22: Γραφήματα Συνάρτησης Rastrigin

5.2.4 Συνάρτηση Rosenbrock (Rosenbrock Function)

Η συνάρτηση Rosenbrock γνωστή και ως συνάρτηση Κοιλάδα ή συνάρτηση Μπανάνα είναι μια δημοφιλής συνάρτηση δοκιμής για τους αλγορίθμους βελτιστοποίησης. Η συνάρτηση είναι μονότροπη (unimodal) δηλαδή διαθέτει μόνο ένα τοπικό ελάχιστο το οποίο είναι και το ολικό και βρίσκεται σε μία στενή παραβολική κοιλάδα. Παρόλο που η εύρεση της κοιλάδας στην οποία βρίσκεται το ελάχιστο είναι εύκολη δεν συμβαίνει το ίδιο και με την εύρεση του ίδιου του ολικού ελαχίστου. Το πεδίο ορισμού της συνάρτησης είναι το $[-5, 10]$, ωστόσο δύναται αναλόγως του αλγορίθμου βελτιστοποίησης και του προβλήματος που αυτός επιλύει να εκτελέσουμε την αξιολόγηση με ένα περιορισμένο πεδίο

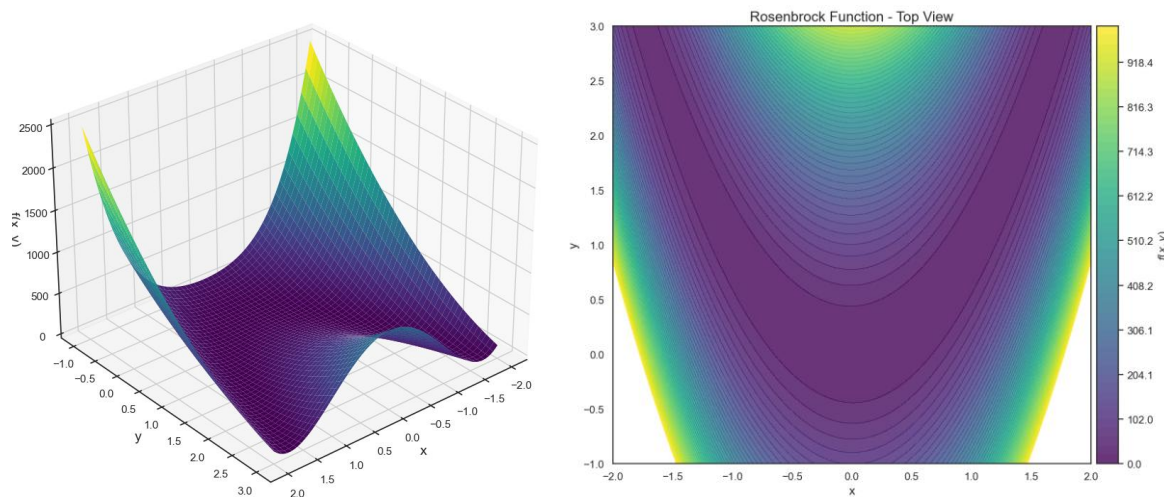
ορισμού και συγκεκριμένα το $[-2.048, 2.048]$. Το ολικό ελάχιστο είναι το $f(1,1,1, \dots, 1) = 0$ και άρα και στις δύο διαστάσεις είναι το $f(1,1) = 0$. [83]

Ο τύπος της συνάρτησης είναι ο εξής:

$$f(x) = \sum_{i=1}^{d-1} [100 \cdot (x_{i+1} - x_i^2)^2 + (x_i - 1)^2] \quad (4.4)$$

όπου d είναι η διάσταση.

Τα γραφήματα της συνάρτησης και της κάτοψης απεικονίζονται παρακάτω:



Εικόνα 23: Γραφήματα Συνάρτησης Rosenbrock

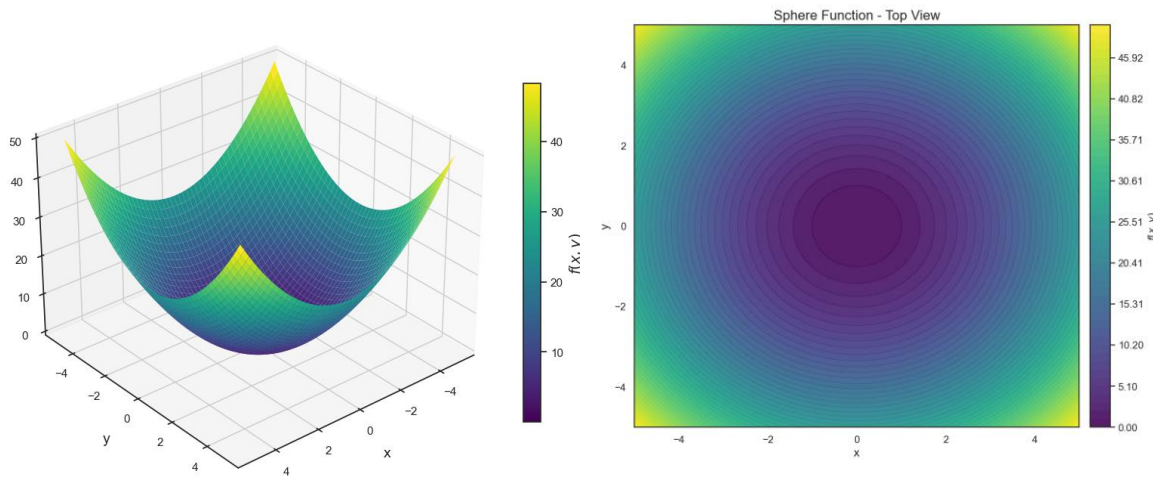
5.2.5 Συνάρτηση Sphere (Sphere Function)

Η συνάρτηση Sphere είναι μία συνάρτηση σχήματος μπουλ η οποία είναι συνεχής, κυρτή και μονότροπη, διαθέτει τόσα τοπικά ελάχιστα όσα και οι διαστάσεις της. Το πεδίο ορισμού της συνάρτησης είναι το $[-5.12, 5.12]$ και το ολικό ελάχιστο είναι το $f(0,0,0, \dots, 0) = 0$ και άρα και στις δύο διαστάσεις θα είναι το $f(0,0) = 0$. [83] Ο τύπος της είναι ο εξής:

$$f(x) = \sum_{i=1}^d x_i^2 \quad (4.5)$$

όπου d είναι η διάσταση.

Τα γραφήματα της συνάρτησης και της κάτοψης απεικονίζονται παρακάτω:

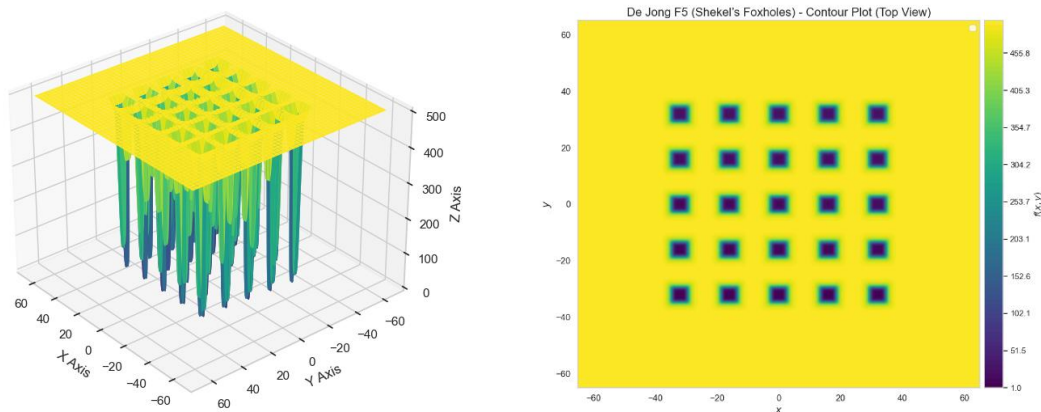


Εικόνα 24: Γραφήματα Συνάρτησης Sphere

5.2.6 Συνάρτηση De Jong No5 ή Shekel's Foxholes (De Jong Function / Shekel's Foxholes Function)

Η πέμπτη συνάρτηση του De Jong είναι μία πολύτροπη συνάρτηση με πολύ απότομες καταβάσεις σε μία επίπεδη επιφάνεια με συνολικά εικοσιπέντε τοπικά ελάχιστα σε κάθε μία από τις εικοσιπέντε απότομες καταβόθρες. Το πεδίο ορισμού της συνάρτησης είναι το $D = \{x \in R^2 \mid x_1, x_2 \in [-65.536, 65.536]\}$. Η συνάρτηση έχει μόνο δύο μεταβλητές και το ολικό ελάχιστο είναι το $f(-32, -32) \approx 1$. [84] [85]

$$f(x_1, x_2) = \left\{ 0.002 + \sum_{i=1}^{25} \frac{1}{[i + (x_1 - a_{1i})^6 + (x_2 - a_{2i})^6]} \right\}^{-1}$$



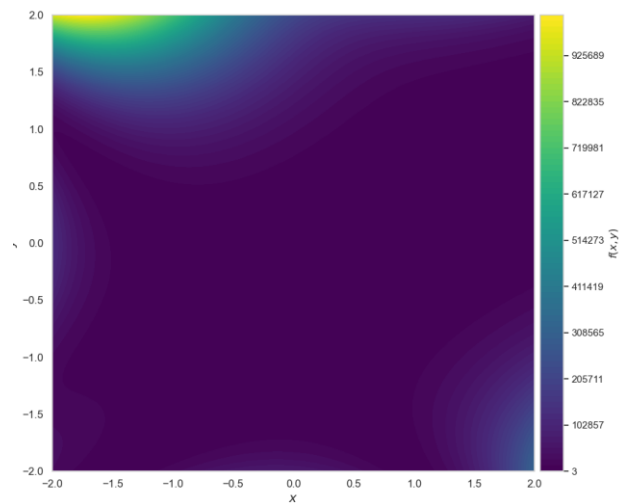
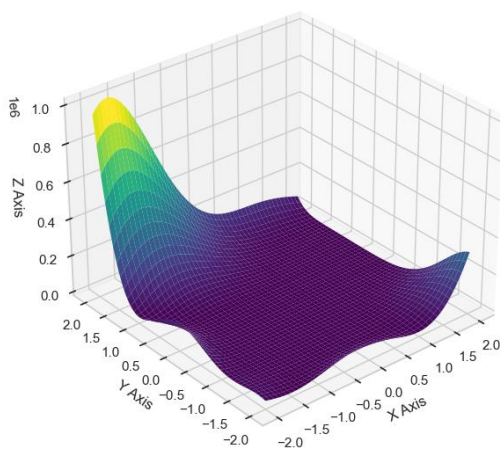
Εικόνα 25: Γραφήματα Συνάρτησης De Jong No5

5.2.7 Συνάρτηση Goldstein-Price (Goldstein-Price Function)

Η συνάρτηση Goldstein-Price έχει αποκλειστικά δύο μεταβλητές και αξιολογείται συνήθως στο τετράγωνο $-2 \leq x_1 \leq 2, -2 \leq x_2 \leq 2$. Η συνάρτηση διαθέτει πολλά τοπικά ελάχιστα και χαρακτηρίζεται από την μεγάλη πεδιάδα που αυτά εντοπίζονται και την μία έντονη ανηφορική πλευρά. Το ολικό βέλτιστό είναι το $f(x) = 3$ για $(x_1, x_2) = (0, -1)$. [86], [87]

Η συνάρτηση έχει τον ακόλουθο τύπο:

$$f(x_1, x_2) = (1 + (x_1 + x_2 + 1)^2 \cdot (19 - 14x_1 + 3x_1^2 - 14x_2 + 6x_1x_2 + 3x_2^2)) \cdot (30 + (2x_1^2 - 3x_2^2) \cdot (18 - 32x_1 + 12x_1^2 + 48x_2 - 36x_1x_2 + 27x_2^2))$$



6. Πειραματική Εφαρμογή και Αποτελέσματα GWO

6.1 Εφαρμογή Αλγορίθμου GWO σε Συναρτήσεις Δοκιμής

Στην παρούσα υποενότητα θα παρουσιαστούν τα αποτελέσματα του αλγορίθμου GWO για τις συναρτήσεις που αναφέρθηκαν στο προηγούμενο κεφάλαιο, σε σχέση με άλλους πολύ γνωστούς μεταερευνητικούς αλγορίθμους και συγκεκριμένα με τον PSO, τον GSA και τον DE.

6.1.1 Σύγκριση Αποτελεσμάτων Με Άλλους Αλγορίθμους

Στους κάτωθι πίνακες παρουσιάζονται ο μέσος όρος και η τυπική απόκλιση των μικρότερων τιμών των καταλληλοτήτων, δηλαδή των καταλληλοτήτων των λύκων άλφα, που υπολογίστηκαν κατά την εκτέλεση τριάντα (30) εκτελέσεων του αλγορίθμου. Το συγκεκριμένο πλήθος εκτελέσεων χρησιμοποιείται συχνά από τους επιστήμονες για την εξαγωγή ασφαλών συμπερασμάτων από την στατιστική ανάλυση των αποτελεσμάτων. Αναλυτικότερα οι τιμές των παραμέτρων του αλγορίθμου απεικονίζονται στον κάτωθι πίνακα:

Όνομα Παραμέτρου	Μεταβλητή	Τιμή
Αριθμός Επαναλήψεων	max_iteration_num	500
Μέγεθος Πληθυσμού	wolves_number	30
Διαστάσεις Προβλήματος ⁸	dimensions_number	30
Ελάχιστη Τυπική Απόκλιση καταλληλότητας	min_fitness_standard_deviation	1e-10

Πίνακας 1 : Τιμές Παραμέτρων Αλγορίθμου

Επισημαίνεται ότι για ευχερέστερη αντιπαραβολή των αποτελεσμάτων της υλοποίησης μας με τα αποτελέσματα που παρουσιάζονται στο [9] διατηρήθηκαν οι αύξοντες αριθμοί των συναρτήσεων δοκιμής.

⁸ Οι διαστάσεις (μεταβλητές προβλήματος) επιλέγονται κατάλληλα στις περιπτώσεις εκείνες των προβλημάτων και των συναρτήσεων δοκιμής που δύναται να διαθέτουν περισσότερες από δύο σε κάθε άλλη περίπτωση όπως στην περίπτωση της Συνάρτησης De Jong F5 και της Goldstein-Price είναι προεπιλεγμένη η τιμή των δύο διαστάσεων.

F	GWO		PSO		GSA		DE	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std
F1	2.59E-10	7.05E-11	0.000136	0.000202	2.53E-16	9.67E-17	8.2E-14	5.9E-14
F5	27.78771	0.55673	96.71832	60.11559	67.54309	62.22534	0	0

Πίνακας 2 : Αποτελέσματα σε Unimodal Συναρτήσεις F1 (Sphere) και F5 (Rosenbrock)

F	GWO		PSO		GSA		DE	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std
F9	2.54E-10	5.02E-10	46.70423	11.62938	25.96841	7.470068	69.2	38.8
F10	1.82E-09	3.57E-10	0.27670	0.50901	0.06208	0.23628	9.7E-08	4.2E-08
F11	3.55E-10	3.99E-10	0.00921	0.00772	27.70154	5.04034	0	0

Πίνακας 3 : Αποτελέσματα σε Multimodal Συναρτήσεις F9 (Rastrigin), F10 (Ackley) και F11 (Griewank)

F	GWO		PSO		GSA		DE	
	Ave	Std	Ave	Std	Ave	Std	Ave	Std
F14	11.54496	2.74643	3.62717	2.56083	5.85984	3.83129	0.998	3.3E-16
F18	3	0.00079	3	1.33E-15	3	4.17E-15	3	2E-15

Πίνακας 4 : Αποτελέσματα σε Multimodal Δύο Διαστάσεων Συναρτήσεις F14 (Shekel's Foxholes – De Jong N5) και F18 (Goldstine-Price)

Από την μελέτη των αποτελεσμάτων που παρατίθενται στους ανωτέρω πίνακες παρατηρούμε τα εξής:

- α) Ο GWO επέδειξε ανταγωνιστική απόδοση και καλή σύγκλιση στην λύση για τις συναρτήσεις Ackley, Sphere, Rastrigin, Griewank και Goldstein-Price συγκρινόμενος με τους PSO και GSA αλλά με μεικτά αποτελέσματα σε σχέση με τον DE ειδικά στις συναρτήσεις Rosenbrock και De Jong N5-Shekel's Foxholes οι οποίες είναι γνωστές για την ιδιαίτερη δυσκολία που παρουσιάζουν.

- β) Επιπροσθέτως παρουσίασε σε αρκετές περιπτώσεις μικρότερη τυπική απόκλιση σε σύγκριση με τον PSO ή τον GSA. Ειδικότερα αυτό συνέβη και στις τρεις multimodal συναρτήσεις όπως φαίνεται στον Πίνακα 3 το οποίο σημαίνει ότι ο αλγόριθμος επέδειξε σταθερότητα στα συγκεκριμένα τοπία. Σε σχέση με τον DE ο GWO επέδειξε ανώτερη απόδοση μόνο στην Rastrigin. Στις υπόλοιπες συναρτήσεις με εξαίρεση τη Goldstein-Price ο DE πέτυχε σημαντικά καλύτερα αποτελέσματα. Για το γεγονός αυτό εκτιμάται ότι ευθύνεται η μορφολογία των συγκεκριμένων συναρτήσεων.

Με βάση τα ανωτέρω συμπεραίνουμε ότι, ο GWO επηρεάζεται από την μορφολογία του τοπίου της συνάρτησης αξιολόγησης, ειδικότερα σε απαιτητικές συναρτήσεις όπως η συνάρτηση Rosenbrock με την στενή κοιλάδα και η συνάρτηση De Jong N5 με τα πολύ βαθιά τοπικά ελάχιστα, η υλοποίηση μας παρουσίασε τάση για στασιμότητα υποδεικνύοντας περιορισμούς στην αποτελεσματική εναλλαγή μεταξύ των φάσεων εξερεύνησης και εκμετάλλευσης σε πολύπλοκα τοπία.

6.1.2 Μελέτη και Ρύθμιση Παραμέτρων

Σε αυτή την υποενότητα θα πειραματιστούμε με τις τιμές των παραμέτρων ώστε να παρατηρήσουμε την επίδραση που έχει κάθε μία από αυτές στην σύγκλιση του αλγορίθμου. Αρχικά θα επιλέξουμε την μονότροπη συνάρτηση Sphere. Σε κάθε ένα από τα παρακάτω πειράματα ο αλγόριθμος εκτελέστηκε τριάντα (30) φορές για εξαγωγή ασφαλών στατιστικών δεδομένων από τα αποτελέσματα.

6.1.2.1 Πρώτο Πείραμα

Σε αυτό το πείραμα θα ξεκινήσουμε με μία αρκετά χαμηλή τιμή στον πληθυσμό των πρακτόρων αναζήτησης προκειμένου να αναδείξουμε την δυσμενή επίδραση που αυτή έχει στην απόδοση του αλγορίθμου.

Όνομα Παραμέτρου	Μεταβλητή	Τιμή
Μέγεθος Πληθυσμού	wolves_number	5
Διαστάσεις Προβλήματος	dimensions_number	30, 2 ⁹
Ελάχιστη Τυπική Απόκλιση καταλληλότητας	min_fitness_standard_deviation	1E-10
Αριθμός Επαναλήψεων	max_iteration_num	500

Πίνακας 5 : Τιμές Παραμέτρων Πρώτου Πειράματος

F	Ελάχιστη Τιμή Fitness	Μέση Ελάχιστη Τιμή Fitness	Διάμεσος Ελάχιστων Τιμών Fitness	Μέσος Όρος Επαναλήψεων Αλγορίθμου	Μέσος Χρόνος Εκτέλεσης GWO (sec)
	Min Fit	Ave Min Fit	Median Min Fit	Ave Iterations	Ave Time
Sphere	1.05E-09	3.04E-08	8.22E-09	460	5.93
Ackley	1.12E-06	1.98E-05	1.21E-05	500	6.08
Griewank	1.62E-09	0.004	4.25E-08	478	7.11
Rastrigin	1.52E-09	3.82E-06	9.48E-09	456	6.02
Rosenbrock	28.10	28.85	28.94	500	6.36
De Jong N5	12.67	12.67	12.67	194	0.53
Goldstein	3	121.21	3.01	500	1.35

Πίνακας 6 : Αποτελέσματα Πρώτου Πειράματος για τις Συναρτήσεις Sphere και Ackley

⁹ Σημειώνεται ότι, οι συναρτήσεις De Jong N5 και Goldstein-Price αξιολογήθηκαν σε δύο διαστάσεις σε όλα τα πειράματα.

6.1.2.2 Δεύτερο Πείραμα

Στο δεύτερο πείραμα αυξάνουμε αρκετά τον πληθυσμό των πρακτόρων στην τιμή τριάντα (30) προκειμένου να παρατηρήσουμε την αναμενόμενη βελτίωση στον υπολογισμό της βέλτιστης τιμής για κάθε συνάρτηση.

Όνομα Παραμέτρου	Μεταβλητή	Τιμή
Μέγεθος Πληθυσμού	wolves_number	30
Διαστάσεις Προβλήματος	dimensions_number	30, 2
Ελάχιστη Τυπική Απόκλιση καταλληλότητας	min_fitness_standard_deviation	1E-10
Αριθμός Επαναλήψεων	max_iteration_num	500

Πίνακας 7 : Τιμές Παραμέτρων Δεύτερου Πειράματος

F	Ελάχιστη Τιμή Fitness	Μέση Ελάχιστη Τιμή Fitness	Διάμεσος Ελάχιστων Τιμών Fitness	Μέσος Όρος Επαναλήψεων Αλγορίθμου	Μέσος Χρόνος Εκτέλεσης GWO (sec)
	Min Fit	Ave Min Fit	Median Min Fit	Ave Iterations	Ave Time
Sphere	1.49E-10	2.52E-10	2.43E-10	290	20.05
Ackley	1.21E-09	1.81E-09	1.83E-09	359	26.91
Griewank	1.25E-10	3.55E-10	2.74E-10	301	22.36
Rastrigin	2.54E-10	5.01E-10	4.40E-10	329	23.83
Rosenbrock	26.44	27.79	28.07	500	36.80
De Jong N5	2.98	11.54	12.67	432	5.07
Goldstein	3	3	3	500	4.73

Πίνακας 8 : Αποτελέσματα Δεύτερου Πειράματος

6.1.2.3 Τρίτο Πείραμα

Όνομα Παραμέτρου	Μεταβλητή	Τιμή
Μέγεθος Πληθυσμού	wolves_number	60
Διαστάσεις Προβλήματος	dimensions_number	30, 2
Ελάχιστη Τυπική Απόκλιση καταλληλότητας	min_fitness_standard_deviation	1E-10
Αριθμός Επαναλήψεων	max_iteration_num	500

Πίνακας 9 : Τιμές Παραμέτρων Τρίτου Πειράματος

F	Ελάχιστη Τιμή Fitness	Μέση Ελάχιστη Τιμή Fitness	Διάμεσος Ελάχιστων Τιμών Fitness	Μέσος Όρος Επαναλήψεων Αλγορίθμου	Μέσος Χρόνος Εκτέλεσης GWO (sec)
	Min Fit	Ave Min Fit	Median Min Fit	Ave Iterations	Ave Time
Sphere	9.46E-11	1.86E-10	1.87E-10	274	36.94
Ackley	7.77E-10	1.01E-09	9.85E-10	323	44.81
Griewank	1.04E-10	2.07E-10	2.08E-10	281	38.42
Rastrigin	1.83E-10	3.07E-10	3.02E-10	307	40.70
Rosenbrock	26.24	27.63	27.62	500	68.09
De Jong N5	2.98	7.22	9.31	495	9.91
Goldstein	3	3	3	500	7.53

Πίνακας 10 : Αποτελέσματα Τρίτου Πειράματος

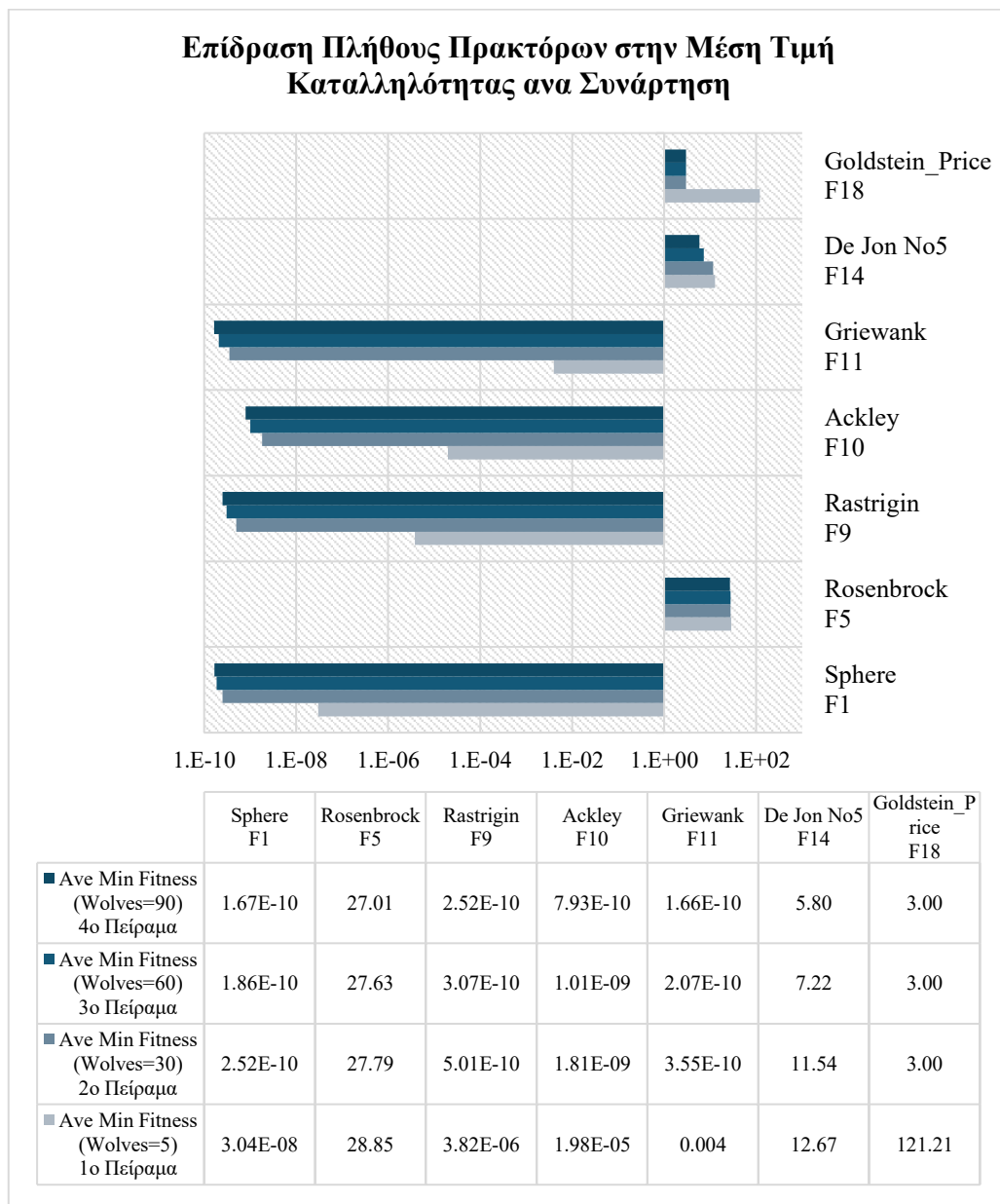
6.1.2.4 Τέταρτο Πείραμα

Όνομα Παραμέτρου	Μεταβλητή	Τιμή
Μέγεθος Πληθυσμού	wolves_number	90
Διαστάσεις Προβλήματος	dimensions_number	30, 2
Ελάχιστη Τυπική Απόκλιση καταλληλότητας	min_fitness_standard_deviation	1E-10
Αριθμός Επαναλήψεων	max_iteration_num	500

Πίνακας 11 : Τιμές Παραμέτρων Τρίτου Πειράματος

F	Ελάχιστη Τιμή Fitness	Μέση Ελάχιστη Τιμή Fitness	Διάμεσος Ελάχιστων Τιμών Fitness	Μέσος Όρος Επαναλήψεων Αλγορίθμου	Μέσος Χρόνος Εκτέλεσης GWO (sec)
	Min Fit	Ave Min Fit	Median Min Fit	Ave Iterations	Ave Time
Sphere	1.06E-10	1.67E-10	1.57E-10	267	55.26
Ackley	5.55E-10	7.93E-10	7.90E-10	310	67.35
Griewank	9.51E-11	1.66E-10	1.55E-10	272	89.65
Rastrigin	1.33E-10	2.52E-10	2.43E-10	302	124.28
Rosenbrock	26.20	27.01	27.19	500	103.90
De Jong N5	2.98	5.80	2.98	497	14.29
Goldstein	3	3	3	500	10.22

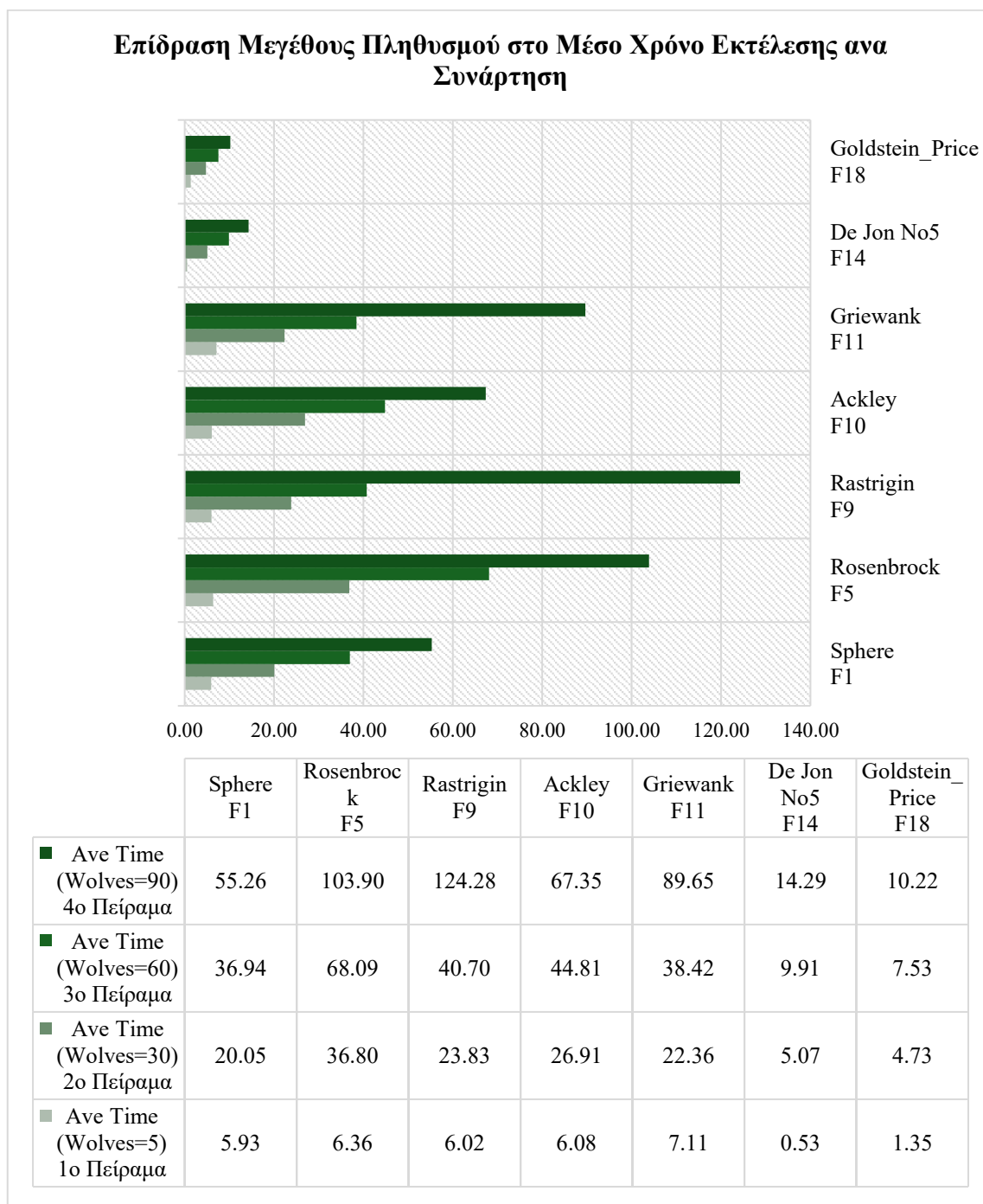
Πίνακας 12 : Αποτελέσματα Τέταρτου Πειράματος



Γράφημα 1 : Γράφημα Απεικόνισης της Επίδρασης του Πλήθους των Πρακτόρων Αναζήτησης

Στο παραπάνω γράφημα απεικονίζεται η επίδραση του μεγέθους του πληθυσμού των λύκων στην ανεύρεση της βέλτιστης λύσης. Παρατηρούμε μια πτωτική τάση στις τιμές της μέσης ελάχιστης καταλληλότητας, που χρησιμοποιήθηκε ως δείκτης για την δημιουργία του γραφήματος, στις περισσότερες συναρτήσεις αξιολόγησης καθώς ο πληθυσμός των πρακτόρων αναζήτησης αυξάνεται σε κάθε ένα από τα τέσσερα πειράματα. Παρατηρώντας το γράφημα γίνεται εμφανές ότι η αύξηση των λύκων από τους πέντε στο πρώτο πείραμα σε τριάντα στο δεύτερο βελτιώνει έντονα την ευρεθείσα λύση, ομοίως παρατηρείται και μια αξιόλογη βελτίωση κατά την μετάβαση στο τρίτο πείραμα με εξήντα πράκτορες

αναζήτησης ενώ στο τέταρτο πείραμα με ενενήντα πράκτορες υπάρχει βελτίωση αλλά αυτή είναι οριακή και συμβαίνει μόνο σε ορισμένες συναρτήσεις. Συμπεραίνουμε λοιπόν ότι ένα μεγαλύτερο πλήθος πρακτόρων αναζήτησης ευνοεί την εύρεση της ολικά βέλτιστης λύσης ή μίας αρκούντως ικανοποιητικής καθώς επιτρέπει καλύτερη εξερεύνηση του χώρου των λύσεων.



Γράφημα 2 : Γράφημα Επίδρασης Μεγέθους Πληθυσμού στον Χρόνο Εκτέλεσης (Sec)

Στο παραπάνω γράφημα απεικονίζονται οι μέσοι χρόνοι εκτέλεσης του αλγορίθμου ανά συνάρτηση για κάθε ένα από τα τέσσερα πειράματα με σταδιακή αύξηση σε κάθε ένα από αυτά του αριθμού των λύκων. Παρατηρούμε ότι στο πρώτο πείραμα με το μικρότερο μέγεθος πληθυσμού καταγράφηκαν και οι μικρότεροι χρόνοι σε όλες τις συναρτήσεις αξιολόγησης σε σχέση με τα υπόλοιπα πειράματα. Επιπροσθέτως παρατηρούμε την δραματική αύξηση του υπολογιστικού χρόνου κατά την μετάβαση από το τρίτο πείραμα με εξήντα πράκτορες στο τέταρτο πείραμα με ενενήντα πράκτορες σε ορισμένες συναρτήσεις (π.χ Rastrigin).

Γενικεύοντας μπορούμε να πούμε ότι η αύξηση του χρόνου μεταξύ των πειραμάτων ήταν σταθερή και σημαντική με την αύξηση του πληθυσμού, ενώ σε ορισμένες συναρτήσεις φαίνεται να ακολουθεί μία γραμμική τάση σε κάποιες άλλες γίνεται αισθητά απότομη σχεδόν εκθετική. Συμπεραίνουμε λοιπόν ότι η αύξηση του πληθυσμού δεν ευνοεί την απόδοση του αλγορίθμου. Ακολούθως θα πειραματιστούμε με τα κριτήρια τερματισμού.

6.1.2.5 Πέμπτο Πείραμα

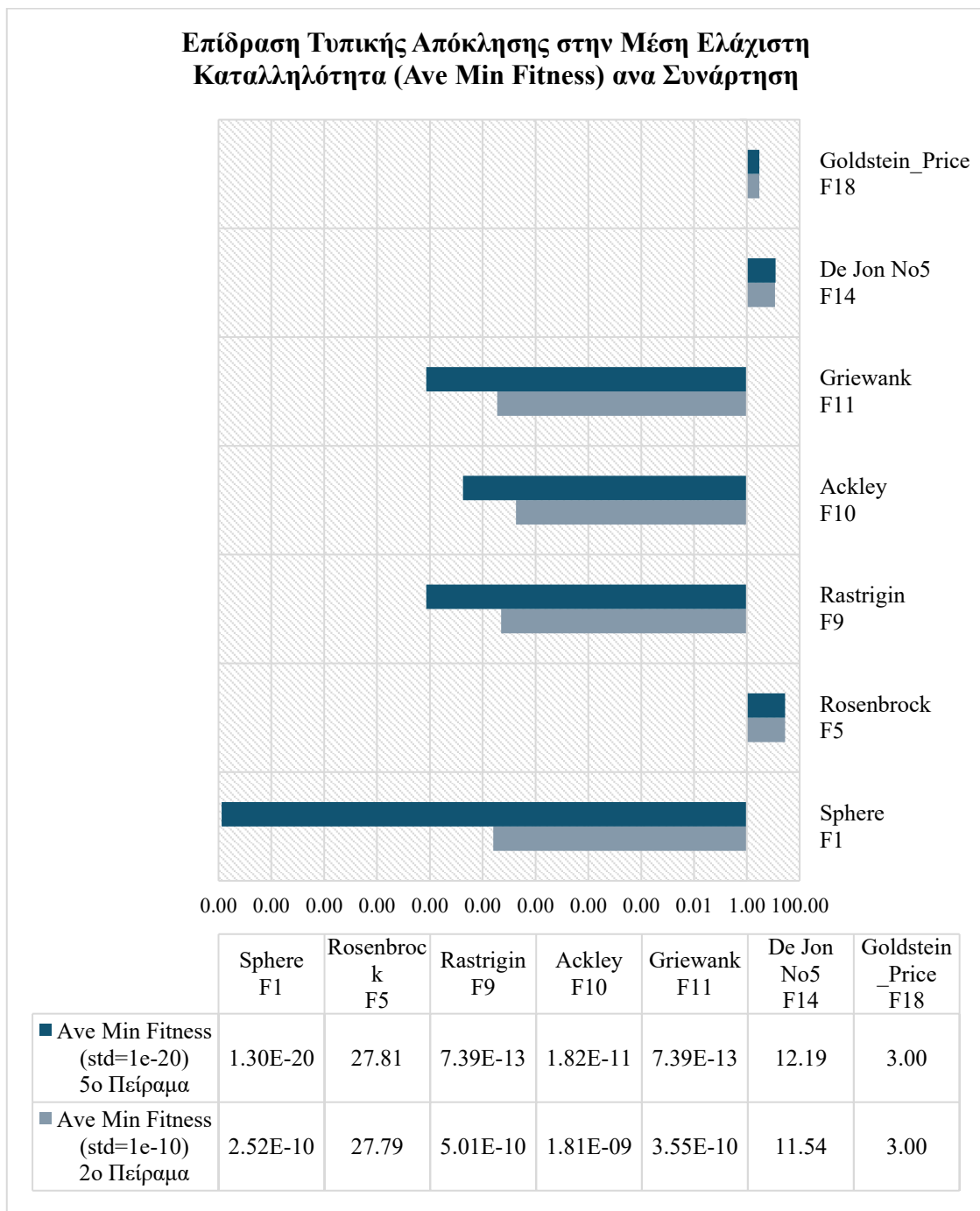
Σε αυτό το πείραμα διατηρούμε τις τιμές των παραμέτρων ίδιες με αυτές του δεύτερου πειράματος και μειώνουμε αισθητά την ελάχιστη τυπική απόκλιση της καταλληλότητας των λύκων. Αναμένουμε αύξηση του χρόνου εκτέλεσης και των εκτελούμενων επαναλήψεων του αλγορίθμου, δεδομένου ότι ο αλγόριθμος θα πρέπει να εκτελέσει λεπτομερέστερη αναζήτηση πλησίον του ολικού βέλτιστου για να το προσεγγίσει με μεγαλύτερη ακρίβεια.

Όνομα Παραμέτρου	Μεταβλητή	Τιμή
Μέγεθος Πληθυσμού	wolves_number	30
Διαστάσεις Προβλήματος	dimensions_number	30, 2
Ελάχιστη Τυπική Απόκλιση καταλληλότητας	min_fitness_standard_deviation	1E-20
Αριθμός Επαναλήψεων	max_iteration_num	500

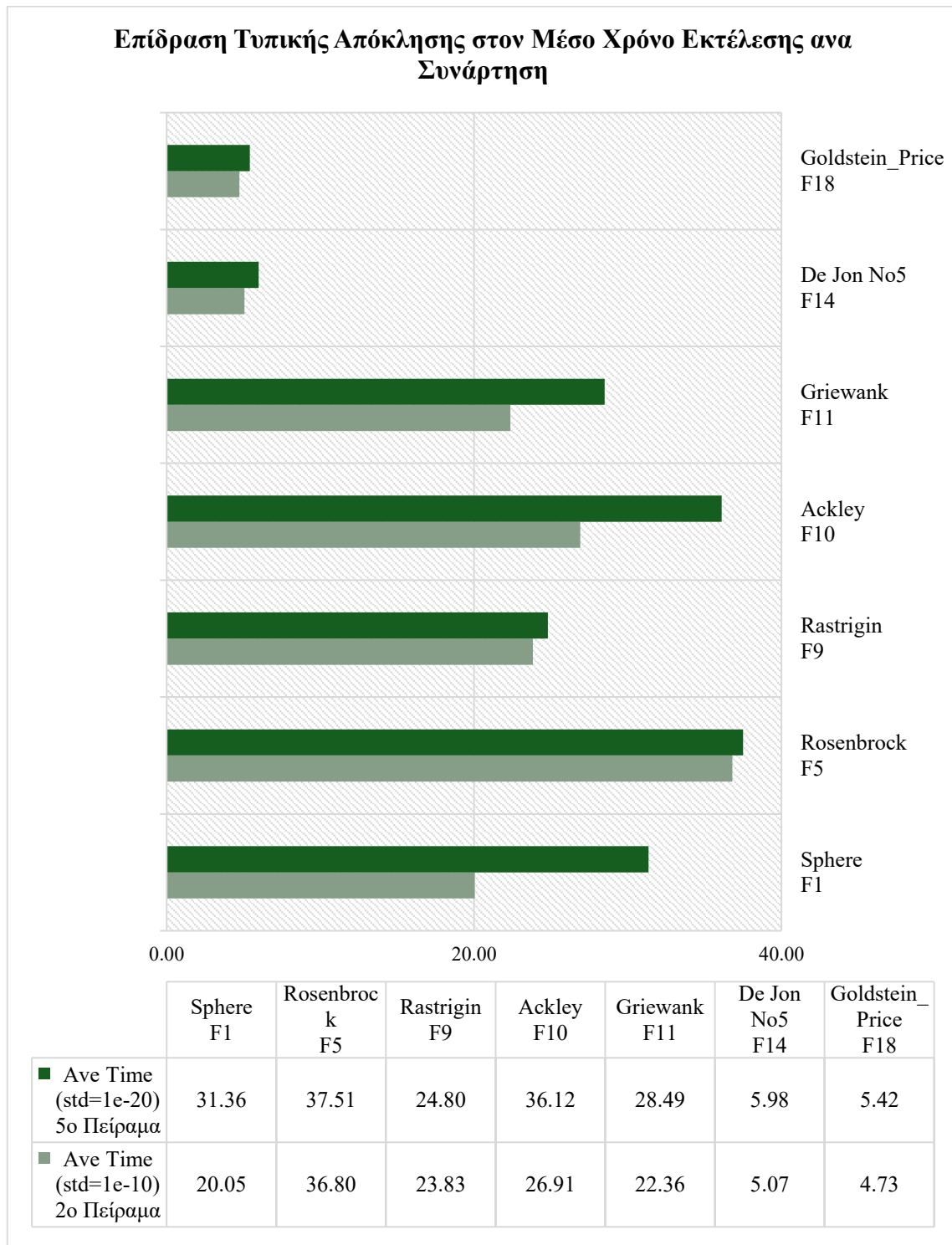
Πίνακας 13 : Τιμές Παραμέτρων Πέμπτου Πειράματος

F	Ελάχιστη Τιμή Fitness	Μέση Ελάχιστη Τιμή Fitness	Διάμεσος Ελάχιστων Τιμών Fitness	Μέσος Όρος Επαναλήψεων Αλγορίθμου	Μέσος Χρόνος Εκτέλεσης GWO (sec)
	Min Fit	Ave Min Fit	Median Min Fit	Ave Iterations	Ave Time
Sphere	5.87E-24	1.30E-20	2.74E-21	500	31.36
Ackley	1.14E-12	1.82E-11	7.73E-11	500	36.12
Griewank	0	0.178	0	404	28.49
Rastrigin	0	7.39E-13	5.68E-14	398	24.80
Rosenbrock	27.17	27.81	28.07	500	37.51
De Jong N5	7.87	12.19	12.67	500	5.98
Goldstein	3	3	3	500	5.42

Πίνακας 14 : Αποτελέσματα Πέμπτου Πειράματος



Γράφημα 3 : Γράφημα Επίδρασης Κριτηρίου Τερματισμού Τυπικής Απόκλισης στην Μέση Ελάχιστη Τιμή Καταλληλότητας



Γράφημα 4 : Γράφημα Επίδρασης Κριτηρίου Τερματισμού Τυπικής Απόκλισης στον χρόνο εκτέλεσης

Όπως παρατηρούμε στον Πίνακα 14 και στα Γραφήματα 3 και 4 η μείωση της τιμής του κριτηρίου τερματισμού του αλγορίθμου οδήγησε σε καλύτερα αποτελέσματα χωρίς ιδιαίτερη χρονική επιβάρυνση. Αυτό οφείλεται εν μέρει στο γεγονός ότι ο αλγόριθμος

περιορίζεται από το μέγιστο πλήθος των 500 επαναλήψεων. Συγκρίνοντας τα αποτελέσματα του Πίνακα 14 με εκείνα του Πίνακα 8, διαπιστώνουμε ότι το αυστηρότερο κριτήριο με τιμή 1E-20 οδήγησε σε σημαντική βελτίωση της ακρίβειας για τις συναρτήσεις Sphere, Ackley και Rastrigin. Αντίθετα, για την συνάρτηση Griewank με το νέο κριτήριο ο αλγόριθμος παρουσίασε σημαντικά χειρότερη απόδοση σύμφωνα με τις μέσες τιμές παρόλο που σε ορισμένες εκτελέσεις υπολόγισε το ακριβές ολικό βέλτιστο. Για τις Συναρτήσεις Rosenbrock, De Jong No5 και Goldstein-Price η απόδοση παρέμεινε ουσιαστικά αμετάβλητη. Λαμβάνοντας υπόψη τα ανωτέρω και παρά την χειρότερη μέση επίδοση στην Griewank, η επιλογή της τιμής 1E-20 κρίνεται ικανοποιητική καθώς προσφέρει βελτιωμένη ακρίβεια στις περισσότερες περιπτώσεις.

Ακολούθως θα εκτελέσουμε ακόμα ένα πείραμα διατηρώντας τις τιμές των παραμέτρων και αυξάνοντας πλέον τις επαναλήψεις του αλγορίθμου από 500 σε 1000 προκειμένου να παρατηρήσουμε την επίδραση στην εύρεση της βέλτιστης λύσης και στον χρόνο εκτέλεσης.

6.1.2.6 Έκτο Πείραμα

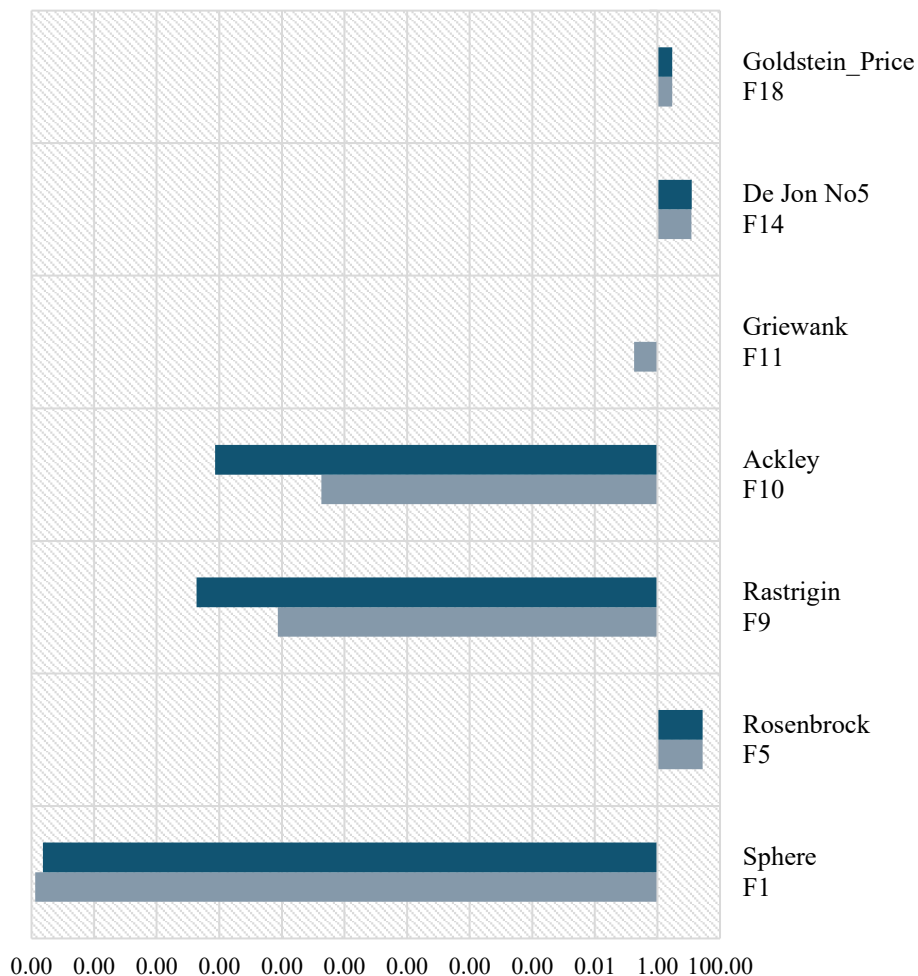
Όνομα Παραμέτρου	Μεταβλητή	Τιμή
Μέγεθος Πληθυσμού	wolves_number	30
Διαστάσεις Προβλήματος	dimensions_number	30, 2
Ελάχιστη Τυπική Απόκλιση καταλληλότητας	min_fitness_standard_deviation	1E-20
Αριθμός Επαναλήψεων	max_iteration_num	1000

Πίνακας 15 : Τιμές Παραμέτρων Έκτου Πειράματος

F	Ελάχιστη Τιμή Fitness	Μέση Ελάχιστη Τιμή Fitness	Διάμεσος Ελάχιστων Τιμών Fitness	Μέσος Όρος Επαναλήψεων Αλγορίθμου	Μέσος Χρόνος Εκτέλεσης GWO (sec)
	Min Fit	Ave Min Fit	Median Min Fit	Ave Iterations	Ave Time
Sphere	1.09E-20	2.30E-20	2.21E-20	552	45.24
Ackley	3.99E-15	7.43E-15	7.55E-15	620	50.52
Griewank	0	0	0	557	48.34
Rastrigin	0	1.89E-15	0	563	46.28
Rosenbrock	27.20	27.70	28.08	1000	82.27
De Jong N5	7.87	12.35	12.67	1000	12.43
Goldstein	3	3	3	1000	10.48

Πίνακας 16 : Αποτελέσματα Έκτου Πειράματος

Επίδραση Επαναλήψεων Αλγορίθμου στην Μέση Ελάχιστη Καταλληλότητα (Ave Min Fitness) ανα Συνάρτηση

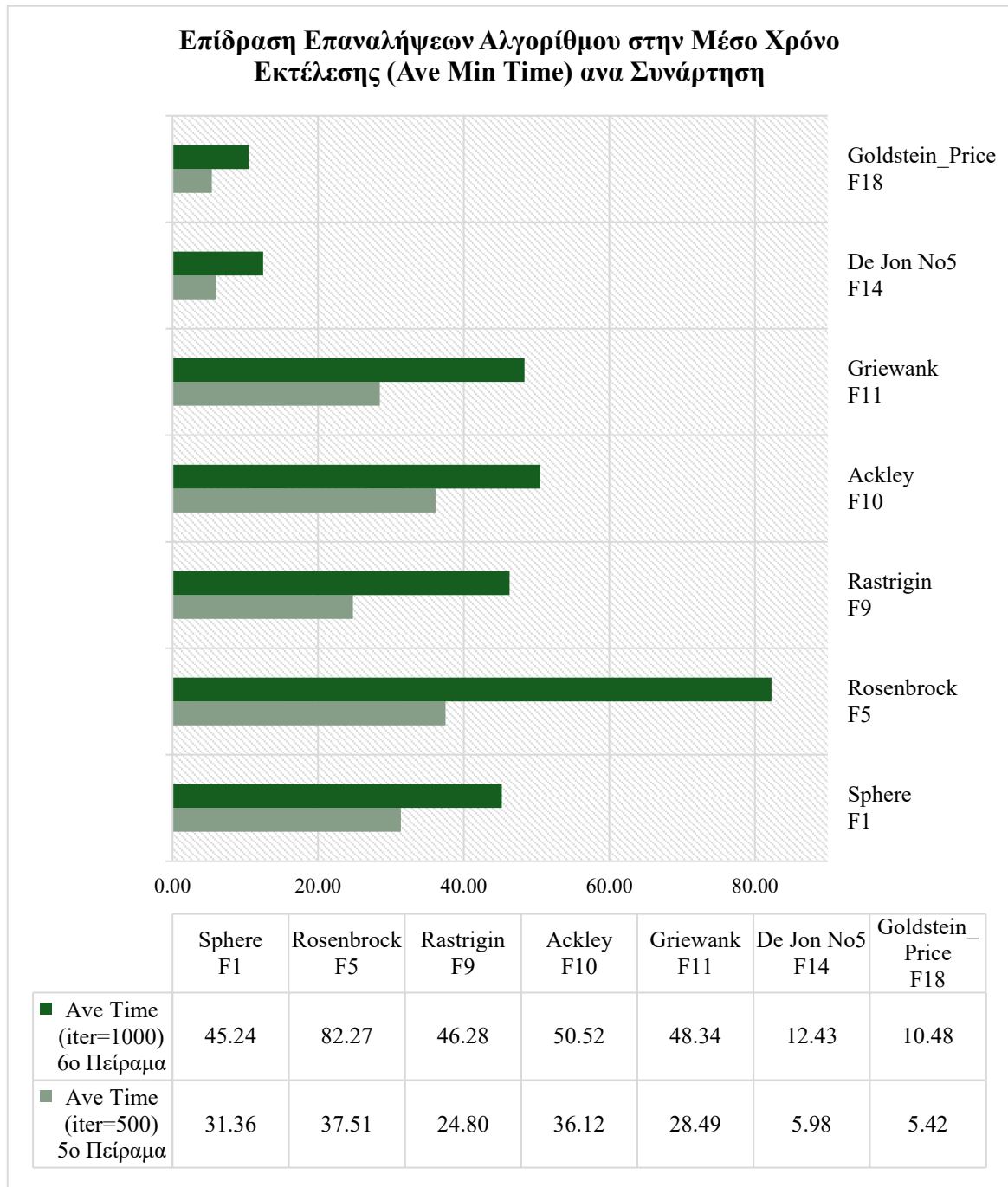


	Sphere F1	Rosenbrock F5	Rastrigin F9	Ackley F10	Griewank F11	De Jon No5 F14	Goldstein_ Price F18
■ Ave Min Fitness (iter=1000) 6ο Πείραμα	2.30E-20	27.70	1.89E-15	7.43E-15	0	12.35	3
■ Ave Min Fitness (iter=500) 5ο Πείραμα	1.30E-20	27.81	7.39E-13	1.82E-11	0.178	12.19	3

Γράφημα 5 : Επίδραση Επαναλήψεων Αλγορίθμου στην Μέση Ελάχιστη Καταλληλότητα ανά Συνάρτηση

Στο ανωτέρω γράφημα παρατηρούμε την βελτίωση της απόδοσης του αλγορίθμου στην εύρεση της βέλτιστης λύσης με την αύξηση των επαναλήψεων και συγκεκριμένα παρατηρούμε ότι στην συνάρτηση Griewank με τις επιπρόσθετες επαναλήψεις ο αλγόριθμος βελτίωσε την λύση και πέτυχε τον υπολογισμό του ολικού βέλτιστου. Επίσης

στην συνάρτηση Rastrigin προσέγγισε αρκετά περισσότερο το ολικό βέλτιστο όπως και στην συνάρτηση Ackley, ενώ εξαίρεση αποτελεί η συνάρτηση Sphere στην οποία παρατηρείται μία πολύ μικρή χειροτέρευση του αποτελέσματος στα δύο πειράματα. Ο λόγος για αυτή την απρόσμενη συμπεριφορά του αλγορίθμου στην Sphere οφείλεται πιθανότητα στην στοχαστικότητα του αλγορίθμου.



Γράφημα 6 : Επίδραση Πλήθος Επαναλήψεων στον Μέσο Χρόνο Εκτέλεσης (Sec)

Στο παραπάνω γράφημα παρατηρούμε την αναμενόμενη αύξηση του χρόνου εκτέλεσης σε σύγκριση με το προηγούμενο πείραμα (5ο Πείραμα). Συγκεκριμένα παρατηρούμε μικρή αύξηση στις συναρτήσεις Sphere, Rastrigin, Griewank και Ackley στις οποίες ο αλγόριθμος δεν εξάντλησε το όριο των 1000 επαναλήψεων αλλά τερμάτισε νωρίτερα (μεταξύ 550 και 620 επαναλήψεων) καθώς πέτυχε τον υπολογισμό της ολικά βέλτιστης λύσης ή μίας εξαιρετικά κοντινής σε αυτήν. Στις υπόλοιπες συναρτήσεις (Rosenbrock, De Jong N5, Goldstein-Price) παρατηρούμε σχεδόν διπλασιασμό του χρόνου καθώς ο αλγόριθμος εξάντλησε και τις 1000 επαναλήψεις κατά την εκτέλεση του.

Από τα ανωτέρω γραφήματα και τους πίνακες των πειραμάτων υπ' αριθμόν 5 και 6 συμπεραίνουμε ότι ένα πλήθος 550-650 επαναλήψεων φαίνεται να αποτελεί μία καλή ισορροπία μεταξύ της επίτευξης υψηλής ακρίβειας για κάποιες συναρτήσεις και περιορισμού του υπολογιστικού κόστους για κάποιες άλλες στις οποίες λόγω τοπολογίας ο αλγόριθμος φαίνεται να συγκλίνει πρόωρα (De Jong No5-Shekel's Foxholes) ή να δυσκολεύεται (Rosenbrock).

6.1.2.7 Έβδομο Πείραμα

Σε αυτό το πείραμα θα αυξήσουμε τον αριθμό των πρακτόρων αναζήτησης σε εξήντα (60) διατηρούμε το βελτιωμένο κριτήριο σύγκλισης σε $1E-20$ και ρυθμίζουμε και τον αριθμό των επαναλήψεων σε πεντακόσιες (500). Αναμένουμε μία βελτιωμένη απόδοση λόγω της αύξησης των πρακτόρων αναζήτησης σε σχέση με τα αποτελέσματα του πειράματος υπ' αριθμόν 5.

Όνομα Παραμέτρου	Μεταβλητή	Τιμή
Μέγεθος Πληθυσμού	wolves_number	60
Διαστάσεις Προβλήματος	dimensions_number	30, 2
Ελάχιστη Τυπική Απόκλιση καταλληλότητας	min_fitness_standard_deviation	$1E-20$
Αριθμός Επαναλήψεων	max_iteration_num	500

Πίνακας 17 : Τιμές Παραμέτρων Έβδομου Πειράματος

F	Ελάχιστη Τιμή Fitness	Μέση Ελάχιστη Τιμή Fitness	Διάμεσος Ελάχιστων Τιμών Fitness	Μέσος Όρος Επαναλήψεων Αλγορίθμου	Μέσος Χρόνος Εκτέλεσης GWO (sec)
	Min Fit	Ave Min Fit	Median Min Fit	Ave Iterations	Ave Time
Sphere	3.50E-20	5.35E-20	5.40E-20	345	54.05
Ackley	4.30E-14	9.03E-14	8.21E-14	472	63.22
Griewank	0	0	0	364	48.65
Rastrigin	0	2.27E-14	0	381	49.11
Rosenbrock	26.23	27.52	27.20	500	77.38
De Jong N5	2.98	6.71	2.98	500	11.09
Goldstein	3	3	3	500	8.88

Πίνακας 18 : Τιμές Αποτελεσμάτων Έβδομου Πειράματος

Από τα αποτελέσματα του παραπάνω πίνακα παρατηρούμε ότι όντως ο αλγόριθμος υπολόγισε με μεγαλύτερη ακρίβεια την λύση, ενώ παράλληλα μειώθηκε και ο αριθμός των εκτελούμενων επαναλήψεων, γεγονός το οποίο σημαίνει ότι ένας μεγαλύτερος πληθυσμός πρακτόρων οδηγεί σε καλύτερη και αποδοτικότερη εξερεύνηση και εκμετάλλευση στον χώρο των λύσεων.

6.1.2.8 Όγδοο Πείραμα

Σε αυτό το πείραμα αυξάνουμε τον αριθμό των επαναλήψεων σε 1000 και διατηρούμε τον αυξημένο πληθυσμό των εξήντα (60) πρακτόρων αναζήτησης. Αναμένουμε ότι ο αλγόριθμος θα οδηγηθεί σε καλύτερες λύσεις αλλά με αυξημένο υπολογιστικό κόστος ειδικά στις συναρτήσεις που παρουσιάζουν δυσκολία λόγω τοπολογίας όπως προαναφέρθηκε.

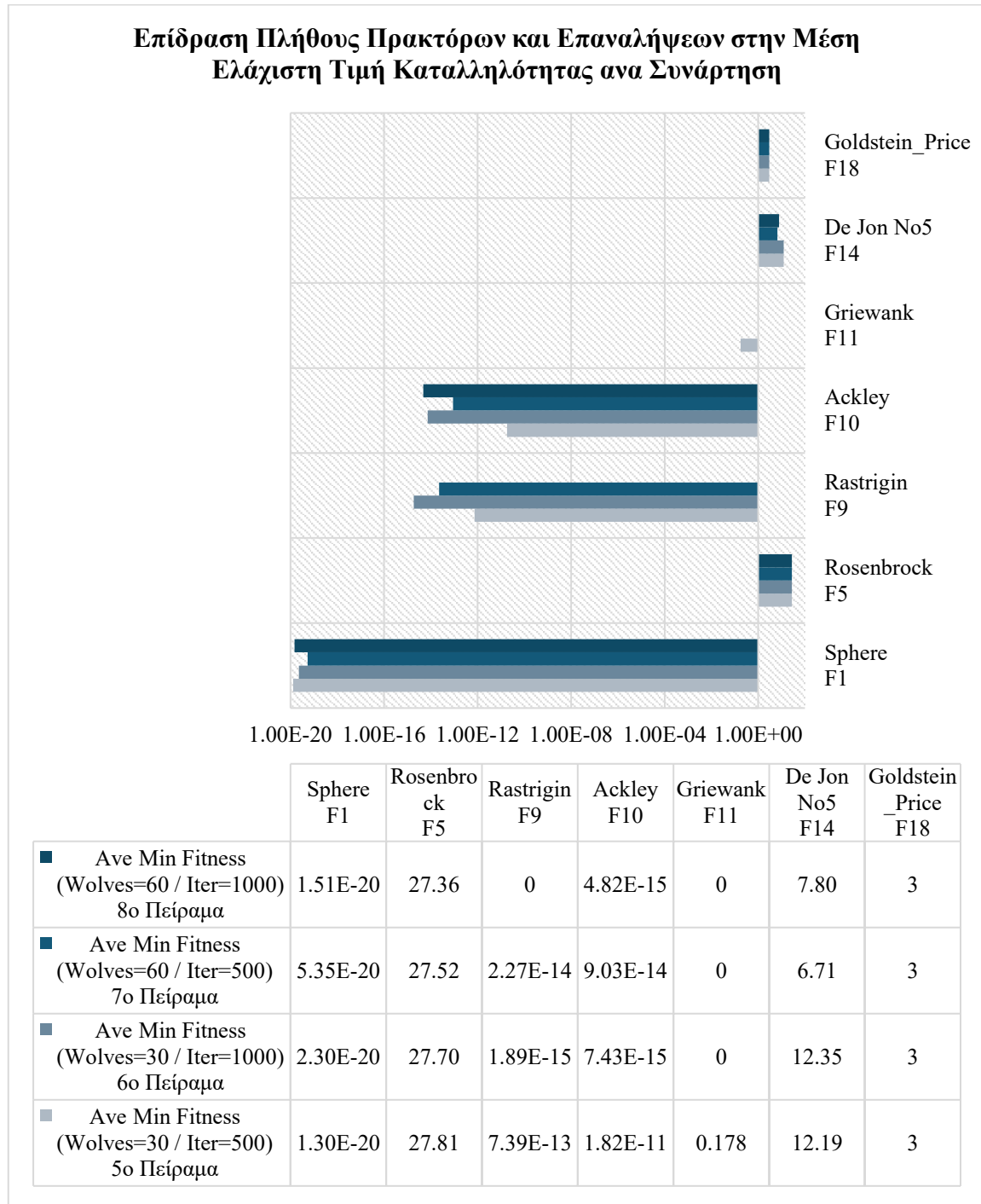
Όνομα Παραμέτρου	Μεταβλητή	Τιμή
Μέγεθος Πληθυσμού	wolves_number	60
Διαστάσεις Προβλήματος	dimensions_number	30, 2
Ελάχιστη Τυπική Απόκλιση καταλληλότητας	min_fitness_standard_deviation	1E-20
Αριθμός Επαναλήψεων	max_iteration_num	1000

Πίνακας 19 : Τιμές Παραμέτρων Όγδοου Πειράματος

F	Ελάχιστη Τιμή Fitness	Μέση Ελάχιστη Τιμή Fitness	Διάμεσος Ελάχιστων Τιμών Fitness	Μέσος Όρος Επαναλήψεων Αλγορίθμου	Μέσος Χρόνος Εκτέλεσης GWO (sec)
	Min Fit	Ave Min Fit	Median Min Fit	Ave Iterations	Ave Time
Sphere	8.85E-21	1.51E-20	1.54E-20	522	69.05
Ackley	3.99E-15	4.82E-15	3.99E-15	587	93.02
Griewank	0	0	0	542	88.01
Rastrigin	0	0	0	550	84.19
Rosenbrock	27.19	27.35	27.20	1000	133.58
De Jong N5	2.98	7.80	10.76	1000	19.81
Goldstein	3	3	3	1000	16.30

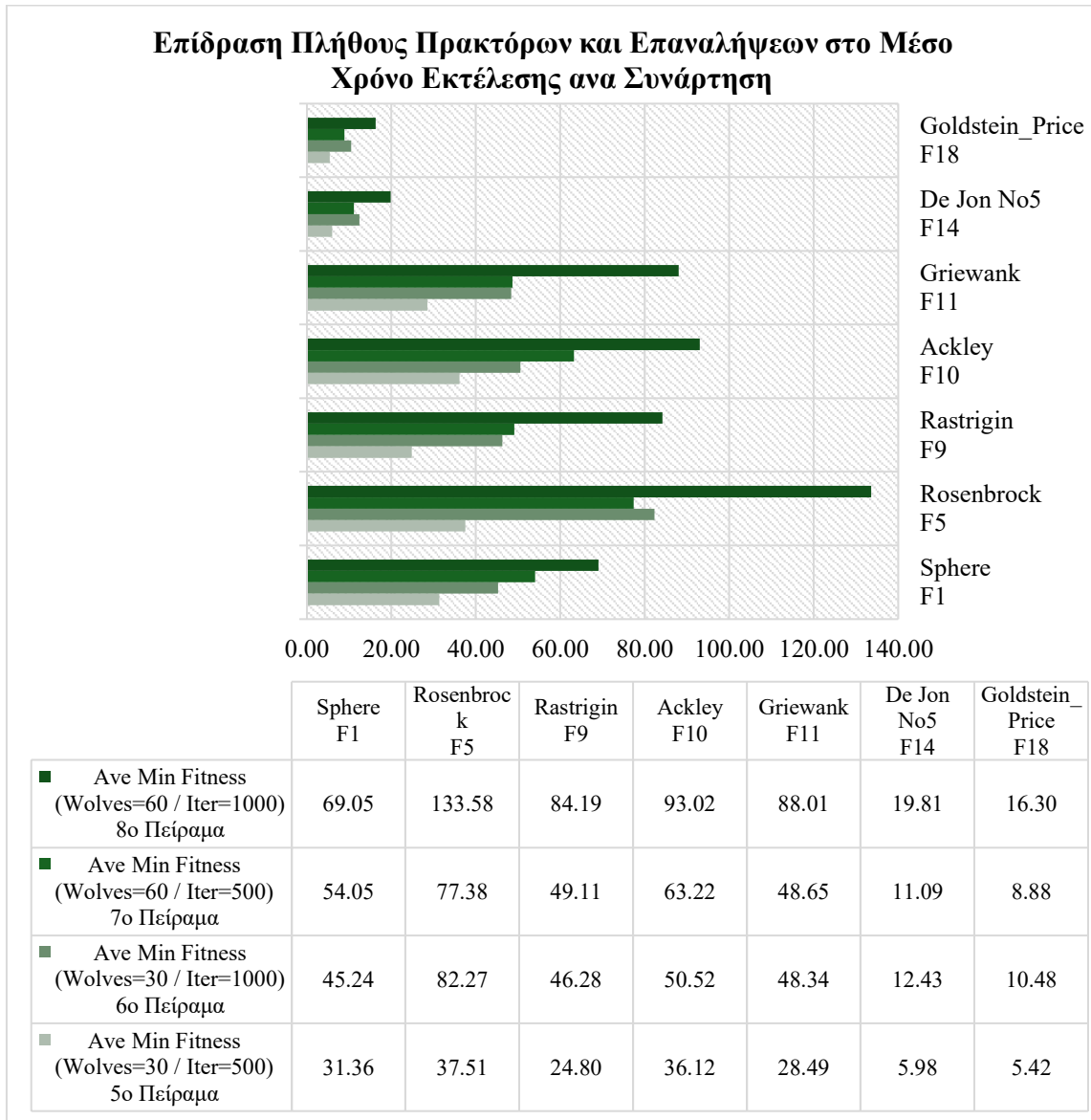
Πίνακας 20 : Τιμές Αποτελεσμάτων Όγδοου Πειράματος

Στον παραπάνω πίνακα παρατηρούμε την αύξηση του χρόνου εκτέλεσης όπως αναμέναμε, ακολούθως θα συγκρίνουμε τα αποτελέσματα των τεσσάρων τελευταίων πειραμάτων ως προς την μέση τιμή καταλληλότητας, τον μέσο χρόνο εκτέλεσης και τον αριθμό των επαναλήψεων.



Γράφημα 7 : Επίδραση Μεγέθους Πληθυσμού και Επαναλήψεων στην Μέση Ελάχιστη Τιμή Καταλληλότητας ανά Συνάρτηση Αξιολόγησης

Στο παραπάνω γράφημα παρατηρούμε ότι τόσο η αύξηση του πληθυσμού των πρακτόρων αναζήτησης όσο και η αύξηση των επαναλήψεων συμβάλλουν στην εύρεση καλύτερων λύσεων. Ωστόσο η επίδραση αυτών των παραμέτρων εξαρτάται από την τοπολογία της συνάρτησης.



Γράφημα 8 : Επίδραση Πλήθους Πρακτόρων και Επαναλήψεων στο Μέσο Χρόνο Εκτέλεσης

Στο ανωτέρω Γράφημα απεικονίζεται ο συμβιβασμός (trade-off) μεταξύ της αναζήτησης καλύτερων λύσεων μέσω της βελτίωσης των τιμών των παραμέτρων και του αντίστοιχου υπολογιστικού κόστους σε χρόνο. Η επιλογή των κατάλληλων τιμών για τις παραμέτρους σχετίζεται άμεσα με την επιθυμητή ισορροπία μεταξύ απόδοσης και διαθέσιμων πόρων.

Τα ανωτέρω διαγράμματα αναδεικνύουν τις πολύπλοκες αλληλεπιδράσεις μεταξύ των παραμέτρων του αλγορίθμου και τον συμβιβασμό μεταξύ ποιότητας λύσης και υπολογιστικού κόστους. Στην επόμενη ενότητα θα αναφερθούμε στα τελικά συμπεράσματα που προέκυψαν τόσο από την συγκριτική αξιολόγηση του GWO με άλλους γνωστούς μεταερευνητικούς αλγορίθμους όσο και από την πειραματική διερεύνηση της απόδοσης του για διάφορες τιμές των παραμέτρων του.

6.2 Συμπεράσματα Αξιολόγησης και Παραμετροποίησης GWO

Από την σύγκριση του GWO με τους αλγορίθμους PSO, GSA και DE στην υποενότητα 6.1. προέκυψαν τα εξής συμπεράσματα:

- Ο GWO επέδειξε ανταγωνιστική απόδοση, συχνά υπερτερώντας του PSO και του GSA επιδεικνύοντας μεγαλύτερη σταθερότητα (μικρότερη τυπική απόκλιση) σε πολύτροπες συναρτήσεις όπως η Ackley, η Rastrigin και η Griewank. Αξιοσημείωτη είναι η απόδοση στην Rastrigin στην οποία ο GWO ξεπέρασε όλους τους συγκρινόμενους αλγορίθμους.
- Ο αλγόριθμος παρουσίασε σημαντικές δυσκολίες στην ανεύρεση ικανοποιητικής λύσης σε απαιτητικά τοπία αναζήτησης όπως η στενή παραβολική κοιλάδα της συνάρτησης Rosenbrock και τα πολύ βαθιά τοπικά ελάχιστα της De Jong No5.

Από τα παραπάνω συμπεραίνουμε η αποτελεσματικότητα του GWO είναι ιδιαίτερα εξαρτώμενη από την μορφολογία του προβλήματος που καλείται να βελτιστοποιήσει.

Επίσης από την πειραματική μελέτη και ρύθμιση των παραμέτρων που πραγματοποιήθηκε στην υποενότητα 6.1.2 εξήχθησαν τα ακόλουθα συμπεράσματα σχετικά με την επίδραση των παραμέτρων στην απόδοση του, αναλυτικότερα:

- Η αύξηση του πληθυσμού των πρακτόρων αναζήτησης από 5 σε 30 και ακολούθως σε 60 βελτιώνει σημαντικά την ποιότητα της λύσης καθώς ενισχύει την ικανότητα εξερεύνησης του χώρου των λύσεων από τον αλγόριθμο. Παρόλα αυτά η βελτίωση της απόδοσης είναι οριακή στην περίπτωση περαιτέρω αύξησης από 60 σε 90 πράκτορες, συγκριτικά με το υπολογιστικό κόστος σε χρόνο, το οποίο αυξάνεται σημαντικά και σχεδόν γραμμικά σε σχέση με την αύξηση του πληθυσμού.

- Επίσης η αύξηση του πληθυσμού από 5 σε 30 και σε 60 φάνηκε να ευνοεί την σύγκλιση του αλγορίθμου καθώς παρατηρήθηκε ταχύτερη σύγκλιση με μειωμένο αριθμό επαναλήψεων σε ορισμένες συναρτήσεις.
- Η μείωση της τιμής της τυπικής απόκλισης από $1E-10$ σε $1E-20$ ομοίως οδήγησε σε γενικές γραμμές σε υψηλότερη ακρίβεια στην τελική λύσης, συγκεκριμένα το γεγονός αυτό παρατηρήθηκε στις συναρτήσεις Ackley, Sphere και Rastrigin. Αντίθετα στην συνάρτηση Griewank παρατηρήθηκε μια απρόσμενη χειροτέρευση της μέσης τιμής της καταλληλότητας της ευρεθείσας λύσης γεγονός που εκτιμάται ότι σχετίζεται με την στοχαστικότητα του αλγορίθμου. Επομένως η τιμή αυτής της παραμέτρου επηρεάζει την εύρεση της βέλτιστης λύσης και θα πρέπει να επιλέγεται προσεκτικά αναλόγως της φύσης του προβλήματος.
- Η αύξηση του αριθμού των επαναλήψεων από 500 σε 1000 επέτρεψε την καλύτερη εξερεύνηση του χώρου των λύσεων και την βελτίωση της απόδοσης του αλγορίθμου. Συγκεκριμένα ο GWO επέστρεψε καλύτερες λύσεις εκμεταλλευόμενος τις επιπλέον επαναλήψεις πετυχαίνοντας σε ορισμένες περιπτώσεις και το ακριβές ολικό βέλτιστο. Στις δύσκολες συναρτήσεις οι επιπλέον επαναλήψεις αύξησαν το υπολογιστικό κόστος χωρίς να βελτιώσουν την ποιότητα της λύσης. Τα πειράματα υπέδειξαν ότι για το δεδομένο σύνολο προβλημάτων ένας αριθμός επαναλήψεων μεταξύ 550-650 αποτελεί έναν καλό συμβιβασμό μεταξύ απόδοσης και υπολογιστικού κόστους σε χρόνο.

Εν κατακλείδι η πειραματική μελέτη επιβεβαίωσε ότι ο GWO είναι ένας ισχυρός μεταερευτικός αλγόριθμος με υψηλή εξερευνητική ικανότητα του χώρου των λύσεων σε ορισμένα τοπία. Η απόδοση του είναι ευαίσθητη τόσο στην φύση του προβλήματος που καλείται να βελτιστοποιήσει όσο και στις τιμές των παραμέτρων του. Η προσεκτική ρύθμιση των παραμέτρων είναι ιδιαίτερα σημαντική λαμβάνοντας υπόψιν και την διαστατικότητα του εκάστοτε προβλήματος προκειμένου να υπάρξει ισορροπία μεταξύ ευρεθείσας λύσης και υπολογιστικού κόστους.

Με βάση τα ευρήματα από τα ανωτέρω πειράματα για προβλήματα περίπου 30 διαστάσεων ένας πληθυσμός 30-60 πρακτόρων και περίπου 600 επαναλήψεων θα αποτελούσαν ένα καλό σημείο εκκίνησης για περαιτέρω βελτιστοποίηση.

7. Βιβλιογραφία

Ακολουθούν οι βιβλιογραφικές αναφορές (πηγές) της Εργασίας.

- [1] Γ. Μπαμπινιώτης, *Λεξικό της Νέας Ελληνικής Γλώσσας*, 2η. Αθήνα: Κέντρο Λεξικολογίας Ε.Π.Ε, 2002.
- [2] “Μεθοδολογία Επίλυσης Προβλημάτων Βελτιστοποίησης.” Accessed: Feb. 11, 2025. [Online]. Available: https://users.uniwa.gr/vmouss/ebooks/optimee/sections/section02_intro_ii.html
- [3] K. Menger, *Das Botenproblem*,” K. Menger, *Kolloquium*, 9, 1932, 12; “Solution of a largescale traveling-salesman problem, . 1932.
- [4] N. Charalambopoulos and A. C. Nearchou, “Ship Routing Using Genetic Algorithms,” *Operations Research Forum*, vol. 2, no. 3, p. 45, 2021, doi: 10.1007/s43069-021-00093-w.
- [5] S. Kim, J. H. Kwak, B. Oh, D. H. Lee, and D. Lee, “An Optimal Routing Algorithm for Unmanned Aerial Vehicles,” *Sensors 2021, Vol. 21, Page 1219*, vol. 21, no. 4, p. 1219, Feb. 2021, doi: 10.3390/S21041219.
- [6] J. Wang *et al.*, “Optimization Method for the Attack Trajectory of Loitering Munitions Based on Damage Pre-assessment,” *J Phys Conf Ser*, vol. 2891, no. 16, p. 162031, Dec. 2024, doi: 10.1088/1742-6596/2891/16/162031.
- [7] X. Sha, “Time Series Stock Price Forecasting Based on Genetic Algorithm (GA)-Long Short-Term Memory Network (LSTM) Optimization,” May 2024, Accessed: Feb. 11, 2025. [Online]. Available: <https://arxiv.org/abs/2405.03151v1>
- [8] N. Paape, J. A. W. M. van Eekelen, and M. A. Reniers, “Simulation-based optimization of a production system topology -- a neural network-assisted genetic algorithm,” Feb. 2024, doi: 10.1080/0951192X.2025.2455631.
- [9] S. Mirjalili, S. M. Mirjalili, and A. Lewis, “Grey Wolf Optimizer,” *Advances in Engineering Software*, vol. 69, pp. 46–61, Mar. 2014, doi: 10.1016/j.adveng-soft.2013.12.007.
- [10] E.-Ghazali. Talbi, *Metaheuristics : from design to implementation*. John Wiley & Sons, 2009.
- [11] “Heuristics : intelligent search strategies for computer problem solving | Semantic Scholar.” Accessed: Feb. 11, 2025. [Online]. Available: <https://www.semanticscholar.org/paper/Heuristics-%3A-intelligent-search-strategies-for-Pearl/8096da6600ad034026c78888139fbf020e09461b>
- [12] M. B. Award, S. I. Gass, M. C. Fu, and D. J. Bartholomew, “Metaheuristics,” *Encyclopedia of Operations Research and Management Science*, pp. 960–970, 2013, doi: 10.1007/978-1-4419-1153-7_1167.
- [13] F. Glover, “Future paths for integer programming and links to artificial intelligence,” *Comput Oper Res*, vol. 13, no. 5, pp. 533–549, Jan. 1986, doi: 10.1016/0305-0548(86)90048-1.
- [14] “[PDF] Modern heuristic optimization techniques :: theory and applications to power systems | Semantic Scholar.” Accessed: Feb. 12, 2025. [Online]. Available:

- <https://www.semanticscholar.org/paper/Modern-heuristic-optimization-techniques-%3A%3A-theory-Lee-El-Sharkawi/f367845c968f726339a4ad2d9455fb75e72cb911>
- [15] P. Agrawal, H. F. Abutarboush, T. Ganesh, and A. W. Mohamed, "Metaheuristic Algorithms on Feature Selection: A Survey of One Decade of Research (2009-2019)," *IEEE Access*, vol. 9, pp. 26766–26791, 2021, doi: 10.1109/ACCESS.2021.3056407.
 - [16] Z. Bousbaa and O. Bencharef, "Metaheuristics for Financial Investment Strategies: Applications Survey," in *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*, IEEE, Jul. 2023, pp. 1–6. doi: 10.1109/ICECCME57830.2023.10253170.
 - [17] F. Glover and G. Kochenberger, *HANDBOOK OF METAHEURISTICS*. New York, Boston, Dordrecht, London, Moscow: Kluwer Academic Publishers, 2003.
 - [18] F. Glover, "Future paths for integer programming and links to artificial intelligence," *Comput Oper Res*, vol. 13, no. 5, pp. 533–549, 1986, doi: 10.1016/0305-0548(86)90048-1.
 - [19] D. H. Wolpert and W. G. Macready, "No free lunch theorems for optimization," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, pp. 67–82, 1997, doi: 10.1109/4235.585893.
 - [20] S. Mirjalili, S. M. Mirjalili, and A. Lewis, "Grey Wolf Optimizer," *Advances in Engineering Software*, vol. 69, pp. 46–61, 2014, doi: 10.1016/j.advengsoft.2013.12.007.
 - [21] M. Gendreau and Potvin Jean-Yves, *Handbook of Metaheuristics*, 3rd ed. Montreal: Springer International Publishing AG, 2019. doi: <https://doi.org/10.1007/978-3-319-91086-4>.
 - [22] M. G. H. Omran and A. Engelbrecht, "Time Complexity of Population-Based Metaheuristics," *MENDEL-Soft Computing Journal*, vol. 29, pp. 2571–3701, 2023, doi: 10.13164/mendel.2023..255.
 - [23] A. Levitin, *Introduction to the Design and Analysis of Algorithms*, 3rd ed. Pearson, 2012.
 - [24] N. S. Jaddi and S. Abdullah, "Global search in single-solution-based metaheuristics," *Data Technologies and Applications*, vol. 54, no. 3, pp. 275–296, Jul. 2020, doi: 10.1108/DTA-07-2019-0115.
 - [25] X. Yu and M. Gen, *Introduction to Evolutionary Algorithms*, vol. 0. in Decision Engineering, vol. 0. London: Springer London, 2010. doi: 10.1007/978-1-84996-129-5.
 - [26] Ι. Διγαλάκης and J. Digalakis, "Παράλληλοι μιμητικοί αλγόριθμοι. Παράλληλοι εξελικτικοί αλγόριθμοι και άλλες τεχνικές," Πανεπιστήμιο Μακεδονίας Οικονομικών και Κοινωνικών Επιστημών, Τμήμα Εφαρμοσμένης Πληροφορικής, Εργαστήριο Παράλληλης και Κατανεμημένης Επεξεργασίας, Θεσσαλονίκη, 2005. doi: 10.12681/eadd/14439.
 - [27] T. Bäck, D. B. Fogel, and Z. Michalewicz, "Handbook of Evolutionary Computation," 1997. [Online]. Available: <https://api.semanticscholar.org/CorpusID:61006071>
 - [28] G. Beni, "From swarm intelligence to swarm robotics," in *Lecture Notes in Computer Science*, Springer Verlag, 2005, pp. 1–9. doi: 10.1007/978-3-540-30552-1_1.
 - [29] G. Beni and J. Wang, "Swarm Intelligence in Cellular Robotic Systems," 1989.
 - [30] R. Eberhart, Y. Shi, and J. Kennedy, *Swarm Intelligence*. San : Morgan Kaufmann, 2001.

- [31] G. Kouziokas, *Swarm Intelligence and Evolutionary Computation*. CRC Press, 2023. doi: 10.1201/9781003247746.
- [32] J. Kennedy and R. Eberhart, "Particle swarm optimization," in *Proceedings of ICNN'95 - International Conference on Neural Networks*, 1995, pp. 1942–1948 vol.4. doi: 10.1109/ICNN.1995.488968.
- [33] G. B. Dantzig and J. H. Ramser, "The Truck Dispatching Problem," *Manage Sci*, vol. 6, no. 1, pp. 80–91, Oct. 1959, doi: 10.1287/mnsc.6.1.80.
- [34] E. Hassanien and Emary Eid, *SWARM INTELLIGENCE Principles, Advances, and Applications*, 1st ed. CRC Press, 2016.
- [35] Y. Tan, *GPU-based Parallel Implementation of Swarm Intelligence Algorithms*, 1st ed. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2016.
- [36] R. Tang, S. Fong, X.-S. Yang, and S. Deb, "Wolf search algorithm with ephemeral memory," in *Seventh International Conference on Digital Information Management (ICDIM 2012)*, IEEE, Aug. 2012, pp. 165–172. doi: 10.1109/ICDIM.2012.6360147.
- [37] S. M. Ross, "Stochastic Processes," 1982. [Online]. Available: <https://api.semanticscholar.org/CorpusID:265774813>
- [38] J. Hull, *Options, Futures, and Other Derivatives*. 1989. [Online]. Available: <https://api.semanticscholar.org/CorpusID:153976727>
- [39] H.-S. Wu and F.-M. Zhang, "Wolf Pack Algorithm for Unconstrained Global Optimization," *Math Probl Eng*, vol. 2014, no. 1, pp. 1–17, 2014, doi: 10.1155/2014/465082.
- [40] A. Vasuki, *Nature-Inspired Optimization Algorithms*. Chapman and Hall/CRC, 2020. doi: 10.1201/9780429289071.
- [41] C. Muro, R. Escobedo, L. Spector, and R. P. Coppinger, "Wolf-pack (*Canis lupus*) hunting strategies emerge from simple rules in computational simulations," *Behavioural Processes*, vol. 88, no. 3, pp. 192–197, Nov. 2011, doi: 10.1016/j.beproc.2011.09.006.
- [42] E. G. Dada, S. B. Joseph, D. O. Oyewola, A. A. Fadele, H. Chiroma, and S. M. Abdulhamid, "Application of Grey Wolf Optimization Algorithm: Recent Trends, Issues, and Possible Horizons," *Gazi University Journal of Science*, vol. 35, no. 2, pp. 485–504, Jun. 2022, doi: 10.35378/GUJS.820885.
- [43] A. Q. H. Badar, "Grey Wolf Optimizer," in *Evolutionary Optimization Algorithms*, Boca Raton: CRC Press, 2021, pp. 165–190. doi: 10.1201/9781003206477-8.
- [44] A. D. Boursianis *et al.*, "Emerging Swarm Intelligence Algorithms and Their Applications in Antenna Design: The GWO, WOA, and SSA Optimizers," *Applied Sciences* 2021, Vol. 11, Page 8330, vol. 11, no. 18, p. 8330, Sep. 2021, doi: 10.3390/APP11188330.
- [45] H. Faris, I. Aljarah, M. A. Al-Betar, and S. Mirjalili, "Grey wolf optimizer: a review of recent variants and applications," *Neural Comput Appl*, vol. 30, no. 2, pp. 413–435, Jul. 2018, doi: 10.1007/s00521-017-3272-5.
- [46] P. Niu, S. Niu, N. liu, and L. Chang, "The defect of the Grey Wolf optimization algorithm and its verification method," *Knowl Based Syst*, vol. 171, pp. 37–43, May 2019, doi: 10.1016/j.knosys.2019.01.018.
- [47] Y. Liu, A. As'array, M. K. Hassan, A. A. Hairuddin, and H. Mohamad, "Review of the grey wolf optimization algorithm: variants and applications," *Neural Comput Appl*, vol. 36, no. 6, pp. 2713–2735, Feb. 2024, doi: 10.1007/s00521-023-09202-8.

- [48] Z.-M. Gao and J. Zhao, "An Improved Grey Wolf Optimization Algorithm with Variable Weights," *Comput Intell Neurosci*, vol. 2019, pp. 1–13, Jun. 2019, doi: 10.1155/2019/2981282.
- [49] P. Hu, S. Chen, H. Huang, G. Zhang, and L. Liu, "Improved Alpha-Guided Grey Wolf Optimizer," *IEEE Access*, vol. 7, pp. 5421–5437, 2019, doi: 10.1109/ACCESS.2018.2889816.
- [50] K. Luo, "Enhanced grey wolf optimizer with a model for dynamically estimating the location of the prey," *Appl Soft Comput*, vol. 77, pp. 225–235, Apr. 2019, doi: 10.1016/j.asoc.2019.01.025.
- [51] S. Gupta and K. Deep, "A novel Random Walk Grey Wolf Optimizer," *Swarm Evol Comput*, vol. 44, pp. 101–112, Feb. 2019, doi: 10.1016/j.swevo.2018.01.001.
- [52] J. Adhikary and S. Acharyya, "Randomized Balanced Grey Wolf Optimizer (RBGWO) for solving real life optimization problems," *Appl Soft Comput*, vol. 117, p. 108429, Mar. 2022, doi: 10.1016/j.asoc.2022.108429.
- [53] S. E. Jørgensen, "Chaos," *Encyclopedia of Ecology, Five-Volume Set*, vol. 1–5, pp. 550–551, Jan. 2008, doi: 10.1016/B978-008045405-4.00148-8.
- [54] M. H. Nadimi-Shahraki, S. Taghian, and S. Mirjalili, "An improved grey wolf optimizer for solving engineering problems," *Expert Syst Appl*, vol. 166, p. 113917, Mar. 2021, doi: 10.1016/j.eswa.2020.113917.
- [55] A. K. Tripathi, K. Sharma, and M. Bala, "A Novel Clustering Method Using Enhanced Grey Wolf Optimizer and MapReduce," *Big Data Research*, vol. 14, pp. 93–100, Dec. 2018, doi: 10.1016/j.bdr.2018.05.002.
- [56] A. V. Chechkin, R. Metzler, J. Klafter, and V. Yu. Gonchar, "Introduction to the Theory of Lévy Flights," in *Anomalous Transport*, Wiley, 2008, pp. 129–162. doi: 10.1002/9783527622979.ch5.
- [57] N. Banerjee and S. Mukhopadhyay, "AP-TLB-IGWO: Adult-pup teaching–learning based interactive grey wolf optimizer for numerical optimization," *Appl Soft Comput*, vol. 124, p. 109000, Jul. 2022, doi: 10.1016/j.asoc.2022.109000.
- [58] Z. Miao, X. Yuan, F. Zhou, X. Qiu, Y. Song, and K. Chen, "Grey wolf optimizer with an enhanced hierarchy and its application to the wireless sensor network coverage optimization problem," *Appl Soft Comput*, vol. 96, p. 106602, Nov. 2020, doi: 10.1016/j.asoc.2020.106602.
- [59] J. Jiang, Z. Zhao, Y. Liu, W. Li, and H. Wang, "DSGWO: An improved grey wolf optimizer with diversity enhanced strategy based on group-stage competition and balance mechanisms," *Knowl Based Syst*, vol. 250, p. 109100, Aug. 2022, doi: 10.1016/j.knosys.2022.109100.
- [60] Z. Yue, S. Zhang, and W. Xiao, "A Novel Hybrid Algorithm Based on Grey Wolf Optimizer and Fireworks Algorithm," *Sensors (Basel)*, vol. 20, 2020, [Online]. Available: <https://api.semanticscholar.org/CorpusID:215771392>
- [61] K. V. Price, "Differential Evolution," 2013, pp. 187–214. doi: 10.1007/978-3-642-30504-7_8.
- [62] J. S. Wang and S. X. Li, "An Improved Grey Wolf Optimizer Based on Differential Evolution and Elimination Mechanism," *Scientific Reports 2019 9:1*, vol. 9, no. 1, pp. 1–21, May 2019, doi: 10.1038/s41598-019-43546-3.
- [63] W. Long, S. Cai, J. Jiao, M. Xu, and T. Wu, "A new hybrid algorithm based on grey wolf optimizer and cuckoo search for parameter extraction of solar photovoltaic

- models,” *Energy Convers Manag*, vol. 203, p. 112243, Jan. 2020, doi: 10.1016/J.ENCONMAN.2019.112243.
- [64] P. J. Gaidhane and M. J. Nigam, “A hybrid grey wolf optimizer and artificial bee colony algorithm for enhancing the performance of complex systems,” *J Comput Sci*, vol. 27, pp. 284–302, Jul. 2018, doi: 10.1016/J.JOCS.2018.06.008.
- [65] H. Xiong, S. Shi, D. Ren, and J. Hu, “A survey of job shop scheduling problem: The types and models,” *Comput Oper Res*, vol. 142, Jun. 2022, doi: 10.1016/j.cor.2022.105731.
- [66] Z. Zhu, X. Zhou, D. Cao, and M. Li, “A shuffled cellular evolutionary grey wolf optimizer for flexible job shop scheduling problem with tree-structure job precedence constraints,” *Appl Soft Comput*, vol. 125, p. 109235, Aug. 2022, doi: 10.1016/J.ASOC.2022.109235.
- [67] “Cellular Automata Machines : A New Environment for Modeling (Scientific Computation),” p. 1987, 1987, Accessed: Sep. 16, 2024. [Online]. Available: https://books.google.com/books/about/Cellular_Automata_Machines.html?hl=el&id=HBIJzrBKUTEC
- [68] S. Mirjalili, S. Saremi, S. M. Mirjalili, and L. D. S. Coelho, “Multi-objective grey wolf optimizer: A novel algorithm for multi-criterion optimization,” *Expert Syst Appl*, vol. 47, pp. 106–119, Apr. 2016, doi: 10.1016/J.ESWA.2015.10.039.
- [69] “Welcome to Python.org.” Accessed: Sep. 25, 2024. [Online]. Available: <https://www.python.org/>
- [70] V. VICHEVA, “Πρόβλεψη τιμών χρονοσειρών με χρήση αλγορίθμων μηχανικής μάθησης και βαθιάς μάθησης.”
- [71] N. Αγγελιδάκης, *Εισαγωγή στον προγραμματισμό με την Python*, 1η. Ηράκλειο, 2015. Accessed: Sep. 25, 2024. [Online]. Available: <http://aggelid.mysch.gr/pythonbook>
- [72] “What is NumPy? — NumPy v2.1 Manual.” Accessed: Nov. 26, 2024. [Online]. Available: <https://numpy.org/doc/stable/user/whatisnumpy.html>
- [73] C. R. Harris *et al.*, “Array programming with NumPy,” *Nature* 2020 585:7825, vol. 585, no. 7825, pp. 357–362, Sep. 2020, doi: 10.1038/s41586-020-2649-2.
- [74] “Statistics (scipy.stats) — SciPy v1.14.1 Manual.” Accessed: Sep. 27, 2024. [Online]. Available: <https://docs.scipy.org/doc/scipy/tutorial/stats.html>
- [75] “Matplotlib: Visualization with Python”, doi: 10.5281/ZENODO.13308876.
- [76] “time — Time access and conversions — Python 3.12.6 documentation.” Accessed: Sep. 27, 2024. [Online]. Available: <https://docs.python.org/3/library/time.html>
- [77] “concurrent.futures — Launching parallel tasks — Python 3.12.7 documentation.” Accessed: Nov. 26, 2024. [Online]. Available: <https://docs.python.org/3.12/library/concurrent.futures.html>
- [78] “Anaconda for Python: Empowering Data Science with Python | Snowflake.” Accessed: Sep. 28, 2024. [Online]. Available: <https://www.snowflake.com/guides/anaconda-python-unleashing-data-science/>
- [79] “What is Google Colaboratory?” Accessed: Sep. 28, 2024. [Online]. Available: <https://deepnote.com/guides/google-cloud/what-is-google-colaboratory>
- [80] “styleguide | Style guides for Google-originated open-source projects.” Accessed: Apr. 15, 2025. [Online]. Available: <https://google.github.io/styleguide/pyguide.html#38-comments-and-docstrings>

- [81] P. Sharma and S. Raju, "Metaheuristic optimization algorithms: a comprehensive overview and classification of benchmark test functions," *Soft comput*, vol. 28, doi: 10.1007/s00500-023-09276-5.
- [82] S. Mirjalili and A. Lewis, "Obstacles and difficulties for robust benchmark problems: A novel penalty-based robust optimisation method," *Inf Sci (N Y)*, vol. 328, pp. 485–509, Jan. 2016, doi: 10.1016/J.INS.2015.08.041.
- [83] S. Surjanovic and D. Bingham, "Virtual Library of Simulation Experiments: Test Functions and Datasets." Accessed: Apr. 05, 2025. [Online]. Available: <https://www.sfu.ca/~ssurjano/optimization.html>
- [84] K. A. De Jong, "Analysis of the behavior of a class of genetic adaptive systems," 1975, Accessed: Apr. 06, 2025. [Online]. Available: <http://deep-blue.lib.umich.edu/handle/2027.42/4507>
- [85] "Alroomi Website - Shekel's Foxholes Function." Accessed: Apr. 06, 2025. [Online]. Available: <https://al-roomi.org/benchmarks/unconstrained/2-dimensions/7-shekel-s-foxholes-function>
- [86] "Goldstein-Price Function." Accessed: Apr. 11, 2025. [Online]. Available: <https://www.sfu.ca/~ssurjano/goldpr.html>
- [87] C. S. Marcin Molga, "Test functions for optimization needs." Accessed: Apr. 11, 2025. [Online]. Available: <https://robertmarks.org/Classes/ENGR5358/Papers/functions.pdf>

Παράρτημα Α «Κώδικας Αλγορίθμου»

A-1 Αρχείο gui.py

```
1. import tkinter as tk
2. from tkinter import ttk, messagebox
3. import ttkbootstrap as tb
4. from ttkbootstrap.constants import *
5.
6. import threading
7. import queue
8. import os
9. import csv
10.
11. from benchmark_function import BenchmarkFunction
12. from gwo import GWO
13.
14. import matplotlib.pyplot as plt
15. from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg, Navigation-
    Toolbar2Tk
16. import numpy as np
17.
18.
19. class GWOApp(tb.Window):
20.
21.     # 1. Αρχικοποιητής
    =====
22.
23.     def __init__(self):
24.         #Αρχικοποίηση του παραθύρου με το θέμα darkly του ttkbootstrap
25.         super().__init__()
26.         # Δηλώνουμε ένα λεξικό που θα αποθηκεύονται τα δεδομένα εισόδου του χρή-
    στη.
27.         self.parameters_dict = {}
28.         #Λίστα με όλα τα διαθέσιμα θέματα του ttkbootstrap
29.         #self.themes =
30.         #[
31.         #     "superhero", "litera", "cosmo", "flatly", "journal", "lumen",
32.         #     "minty", "pulse", "sandstone", "united", "yeti", "cyborg",
33.         #     "darkly", "solar", "vapor"
34.         #]
35.         self.themes = ["minty", "cosmo", "sandstone", "united", "yeti", "superhero", "so-
    lar", "vapor", "cyborg"]
36.         self.dark_ttk_themes = ["superhero", "solar", "vapor", "cyborg"]
37.         # Το αρχικό Θέμα της εφαρμογής μας
38.         initial_theme = "cyborg"
39.         # Αρχικά Θέτουμε το background των γραφημάτων να είναι Σκουρόχρωμο
```

```

40.     self.plot_dark_background_var = tk.BooleanVar(value = True)
41.     # Καλούμε την private μέθοδο για αρχικοποίηση των χρωμάτων φόντου των
γραφημάτων.
42.     self._update_plot_bg_colors()
43.
44.     self.plot_update_counter = 0
45.     self.style.theme_use(initial_theme)
46.
47.     # --- Μεταβλητές για τα control Buttons των γραφημάτων ---
48.     # Μεταβλητή για το Toggle Button του γραφήματος Σύγκλισης Καταλληλότη-
τας.
49.     self.enable_convergence_plot_var = tk.BooleanVar(value=True)
50.     # Για τα RadioButtons (3d_view και top_view)
51.     self.active_spatial_plot_var = tk.StringVar(value="3d_view")
52.     # Ενεργοποίηση της ενημέρωσης των γραφημάτων του χώρου αναζήτησης και
των πρακτόρων (λύκων).
53.     self.enable_spatial_live_update_var = tk.BooleanVar(value=True)
54.
55.     self.title("GWO Algorithm Simulator")
56.     self.geometry("1350x850")
57.
58.     self.benchmark_functions_name_list = ["ackley", "griewank", "rastrigin", "ros-
enbrock", "sphere", "dejongF5_foxholes", "goldstein-price"]
59.     self.fixed_dim_functions = ["dejongF5_foxholes", "goldstein-price"]
60.
61.     # --- Μεταβλητές για την πολυνηματική επεξεργασία και την επικοινωνία με-
ταξύ του νήματος του GUI και του GWO ---
62.     self.gwo_thread = None # Νήμα εκτέλεσης GWO
63.     self.stop_event = threading.Event() # Event για την διακοπή εκτέλεσης του νή-
ματος
64.     self.gui_queue = queue.Queue() # Thread Safe δομή (ουρά) για την αποστολή
μνμ απο το νήμα του GWO στο νήμα του GUI
65.     self.parameters_dict = {} # Λεξικό για την αποθήκευση των τιμών των παραμέ-
τρων απο τα πεδία εισόδου του GUI
66.
67.     # --- Μεταβλητές και Ρυθμίσεις Γραφημάτων ---
68.     # Λεξικό για την αποθήκευση των στοιχείων της κάθε καρτέλας (fig, ax, canvas)
69.     self.plot_tabs_dict = {}
70.     self.plot_configs_list = [
71.         {"key": "3d_view", "title": "3D View", "projection": "3d"},
72.         {"key": "top_view", "title": "Top View", "projection": None},
73.         {"key": "convergence", "title": "Convergence Curve", "projection": None}
74.     ]
75.     self.convergence_data = {"iterations": [], "fitness": []}
76.
77.     # Το τρέχον αντικείμενο benchmark για το οποίο έχει σχεδιαστεί η συνάρτηση
αξιολόγησης
78.     self.current_benchmark_object_for_spatial_plot = None

```

```

79.      # Μεταβλητή για να κρατάμε το αντικείμενο της γραφικής παράστασης 3D
View
80.      self.plotted_3d_surface = None
81.      # Ομοίως για την γραφική παράσταση top_view
82.      self.plotted_top_view_contour = None
83.      # Βοηθητική μεταβλητή για το colorbar της Top View
84.      self.plotted_top_view_colorbar = None
85.      # Βοητική μεταβλητή για τις θέσεις των Λύκων στο 3D Διάγραμμα
86.      self.scatter_3d_wolves = None
87.      # Βοηθητική Μεταβλητή για τις θέσεις των Λύκων στο Top View
88.      self.scatter_top_view_wolves = None
89.      # Μεταβλητή για την άθροιση του χρόνου εκτέλεσης σε κάθε RUN
90.      self.cumulative_execution_time = 0.0
91.
92.      # Καλούμε την μέθοδο create_widgets() για την σχεδίαση των widgets στο GUI.
93.      self.create_widgets()
94.      # Εκτέλεση της process_gui_queue() κάθε 100ms για έλεγχο μνμ εισερχόμενων
απο το νήμα του GWO
95.      self.after(100, self.process_gui_queue)
96.
97.      # Παρεμβαίνουμε στο κλείσιμο της εφαρμογής απο το ΛΣ όταν ο χρήστης πα-
τάει το 'X' του παραθύρου,
98.      # εκτελώντας την δική μας συνάρτηση on_closing()
99.      self.protocol("WM_DELETE_WINDOW", self.on_closing)
100.
101.      # 2. Μέθοδοι Δημιουργίας Κύριων Τμημάτων του GUI (Layout)
=====
102.
103.      def create_widgets(self):
104.          """
105.              Η μέθοδος δημιουργεί όλα τα κύρια widgets (Frames, Input fields, Labels,
Buttons, Plots) και τα
106.              τοποθετεί μέσα στο κύριο παράθυρο της εφαρμογής.
107.              Δημιουργούνται:
108.              1 πλαίσιο για το combobox Και το toggle button για την επιλογή Σκούρου
Θέματος και Φόντου
109.              Γραφημάτων,
110.              1 panedWindow που χωρίζει το παράθυρο σε δεξί και αριστερό μέρος,
111.              1 left pane που περιέχει τα πεδία εισόδου των παραμέτρων και τα toggle but-
tons και spinbox για
112.              τον έλεγχο της εμφάνισης των γραφημάτων καθώς και κουμπιά έναρξης και
παύσης του αλγορίθμου.
113.              1 right pane που περιέχει ένα Notebook με καρτέλες για καθε γραφική παρά-
σταση.
114.          """
115.          # Δημιουργούμε ένα πλαίσιο container που θα μπει στην κορυφή του κυρίως
παραθύρου.
116.          top_frame = tb.Frame(self, padding=(10, 10, 10, 0))
117.          # Το τοποθετούμε στο κυρίως παράθυρο, στο πάνω μέρος και καταλαμβάνει

```

```
118. # όλο το μήκος του παραθύρου (fill=X).
119. top_frame.pack(fill=X, side=TOP)
120. # Δημιουργούνται τα widget για την επιλογή του θέματος και του φόντου γρα-
    φήματος
121. self.create_theme_chooser_frame(top_frame)
122.
123. # Δημιουργία ενός αντικειμένου PanedWindow() στο κυρίως παράθυρο της ε-
    φαρμογής
124. # που το χωρίζει σε δύο κάθετα τμήματα αριστερό και δεξί μεταβλητού μήκους.
125. main_paned_window = ttk.PanedWindow(self, orient=HORIZONTAL)
126. # Το τοποθετούμε στο παράθυρο και το ρυθμίζουμε για πλήρη κάλυψη της πε-
    ριοχής, αυτόματη προσαρμογή στο παράθυρο.
127. main_paned_window.pack(fill=BOTH, expand=True, padx=10, pady=10)
128.
129. # Δημιουργούμε ένα Frame για την ομαδοποίηση widgets.
130. left_frame = tk.Frame(main_paned_window, padding=10)
131. # Το τοποθετούμε μέσα στο paned window.
132. main_paned_window.add(left_frame)
133.
134. # Ομοίως δημιουργούμε ένα ακόμα Frame και το τοποθετούμε και αυτό στο
    δεξί panedWindow.
135. right_frame = tk.Frame(main_paned_window, padding=10)
136. main_paned_window.add(right_frame)
137.
138. # --- Καλούμε μεθόδους για την δημιουργία των περιεχομένων των Frames
139. # Δημιουργία των πεδίων εισόδου των παραμέτρων του αλγορίθμου.
140. self.create_parameters_frame(left_frame)
141. # Δημιουργία των toggle buttons για την εμφάνιση των γραφημάτων και των
    Control Buttons (START-STOP)
142. self.create_plot_control_options_frame(left_frame)
143. # Εμφάνιση της μπάρας προόδου της διαδικασίας βελτιστοποίησης
144. self.create_progress_frame(left_frame)
145. # Δημιουργία του Frame για την εμφάνιση μυνημάτων στον Χρήστη.
146. self.create_log_frame(left_frame)
147. # Δημιουργία του Notebook στο δεξί μέρος του παραθύρου με τις
148. # καρτέλες των γραφημάτων.
149. self.create_plots_frame(right_frame)
150. # Εφαρμογή των χρωμάτων του επιλεγθέντος θέματος στα γραφήματα
151. # για καλύτερο UI/UX.
152. self.on_theme_select(update_plots=True)
153.
154.
155. def create_theme_chooser_frame(self, parent):
156.     """
157.     Δημιουργούμε ένα πλαίσιο και τοποθετούμε ένα Label και ένα combobox
158.     για την επιλογή ενός θέματος απο τα διαθέσιμα της λίστας.
159.
160.     Args:
161.     parent (tk.widget): Περνιέται ως παράμετρος το γονικό widget μέσα στο
```

```

162.         οποίο θα δημιουργηθεί το LabelFrame των widgets για το Log Frame.
163.         """
164.         # Δημιουργούμε ένα εξωτερικό πλαίσιο για τα δύο widgets επιλογής θέματος
165.         # και χρώματος φόντου γραφημάτων.
166.         theme_outer_frame = tk.Frame(parent)
167.         # Τοποθετούμε το Frame στην δεξιά πλευρά του γονικού παραθύρου
168.         # για να είναι ευδιάκριτα από τον Χρήστη
169.         theme_outer_frame.pack(side=RIGHT, fill=X, padx=5)
170.
171.         # Δημιουργούμε το Frame που θα φιλοξενήσει το dropdown επιλογής θέματος
172.         # και το αγκιστρώνουμε στο εξωτερικό Frame
173.         theme_select_frame = tk.Frame(theme_outer_frame)
174.         theme_select_frame.pack(side=LEFT, padx=(0,10))
175.         tb.Label(theme_select_frame, text="App Theme:").pack(side=LEFT,
162.         padx=(0,5))
176.
177.         # Δημιουργία του Combobox για την επιλογή θέματος.
178.         self.theme_combo = ttk.Combobox(theme_select_frame,
179.                                         values=self.themes,
180.                                         state="readonly",
181.                                         width=15)
182.         self.theme_combo.pack(side=LEFT)
183.         # Ορίζει την αρχική επιλεγμένη τιμή του combobox να είναι το ενεργό τρέχον
184.         θέμα
185.         self.theme_combo.set(self.style.theme.name)
186.         # Όταν ο χρήστης κάνει μια επιλογή από το combobox καλείται η μέθοδος
187.         on_theme_select().
188.         self.theme_combo.bind("<<ComboboxSelected>>", self.on_theme_select)
189.
190.         # Δημιουργούμε ακόμη ένα εσωτερικό Frame που θα βάλουμε ένα checkbox
191.         για
192.         # την επιλογή σκουρόχρωμου φόντου στα γραφήματα
193.         plot_style_frame = tk.Frame(theme_outer_frame)
194.         plot_style_frame.pack(side=LEFT)
195.         self.plot_bg_check = tk.Checkbutton(
196.             plot_style_frame,
197.             text = "Dark Plot Background",
198.             variable = self.plot_dark_background_var,
199.             bootstyle = "primary-round-toggle",
200.             command = self.on_plot_background_toggle
201.         )
202.         self.plot_bg_check.pack(side=LEFT, padx=5)
203.
204.         def create_parameters_frame(self, parent):
205.             """
206.             Δημιουργούμε ένα πλαίσιο για να ομαδοποιήσουμε τα widgets πεδίων εισό-
207.             δων των παραμέτρων
208.
209.             Args:

```

```
206.     parent(tk.widget): Περνιέται ως παράμετρος το γονικό widget μέσα στο ο-  
ποίο θα δημιουργηθεί  
207.     το LabelFrame των παραμέτρων.  
208.     ""  
209.     # Δημιουργούμε ένα LabelFrame, ένα πλαίσιο με ετικέτα για την φιλοξενία.  
210.     parameters_frame = tk.LabelFrame(parent, text="Parameters", padding=10,  
bootstyle=PRIMARY)  
211.     # Τοποθετούμε το Frame στο γονικό widget που είναι το left_frame το οποίο  
και θα πιάσει όλο το  
212.     # πλάτος του left_frame (fill=X), pady=(0,10) σημαίνει εσωτερικό κενό πάνω 0  
και κάτω 10.  
213.     parameters_frame.pack(fill=X, pady=(0,10))  
214.  
215.     # --- Widget 1: Επιλογή Συνάρτησης (Combobox).  
216.     tb.Label(parameters_frame, text="Benchmark Function: ").grid(row=0, col-  
umn=0, padx=5, pady=5, sticky=W)  
217.     # Αποθηκεύουμε στο ζεύγος του λεξικού με key=benchmark την τιμή μίας με-  
ταβλητή τύπου StringVar  
218.     # η οποία ακούει για τυχόν αλλαγές.  
219.     self.parameters_dict["benchmark"] = tk.StringVar()  
220.  
221.     # Δημιουργούμε ένα combobox για να επιλέγει ο χρήστης συνάρτηση αξιολό-  
γησης από μία λίστα  
222.     # και να αποθηκεύεται στην μεταβλητή StringVar και άρα στο ζεύγος του λεξι-  
κού με key=benchmark.  
223.     self.benchmark_combobox = ttk.Combobox(  
224.         parameters_frame, # Τοποθετούμε το combobox στο Frame των παραμέτρων  
225.         textvariable=self.parameters_dict["benchmark"], # Συνδέουμε το widget με  
την μεταβλητή self.parameters_dict["benchmark"]  
226.         values=self.benchmark_functions_name_list, # Περνάμε ως όρισμα το πεδίο  
της κλάσης που είναι μία λίστα συναρτήσεων αξιολόγησης  
227.         state="readonly", # Δεν μπορεί ο Χρήστης να γράψει στο combobox  
228.         width=25  
229.     )  
230.     # Προσθέτουμε ένα Placeholder  
231.     self.benchmark_combobox.set("Select a function...")  
232.     # Τοποθετούμε το combobox στο πλέγμα στην γραμμή 0, στήλη 1 και κολλάει  
από την Ανατολική Πλευρά μέχρι την Δυτική (EAST-WEST=EW).  
233.     self.benchmark_combobox.grid(row=0, column=1, padx=5, pady=5,  
sticky=EW)  
234.     # Με την κάτωθι εντολή δημιουργούμε έναν event listener στο combobox.  
235.     self.benchmark_combobox.bind("<<ComboboxSelected>>", self.on_bench-  
mark_select)  
236.  
237.     # --- Widget 2: Πεδίο Κειμένου για τις Διαστάσεις  
238.     tb.Label(parameters_frame, text="Dimensions:").grid(row=1, column=0,  
padx=5, pady=5, sticky=W)  
239.     # Η μεταβλητή StringVar() συνδέεται με το widget και ακούει για τιμές  
εισόδου από τον Χρήστη
```

```

240.     self.parameters_dict["dimensions"] = tk.StringVar(value="2")
241.     # Δημιουργούμε το Πεδίο Εισόδου και το συνδέουμε με την μεταβλητή self.pa-
parameters_dict["dimensions"]
242.     self.dimensions_entry = tb.Entry(parameters_frame, textvariable=self.parame-
ters_dict["dimensions"], state="disabled", width=28)
243.     self.dimensions_entry.grid(row=1, column=1, padx=5, pady=5, sticky=EW)
244.
245.     # --- Widget 3: Πεδίου Κειμένου για το πλήθος των Λύκων
246.     tb.Label(parameters_frame, text="Wolves Number:").grid(row=2, column=0,
padx=5, pady=5, sticky=W)
247.     self.parameters_dict["wolves"] = tk.StringVar(value="30")
248.     #Ομοίως πεδίο εισόδου για την επιλογή πρακτόρων αναζήτησης, σύνδεση με
μεταβλητή self.parameters_dict["wolves"]
249.     self.wolves_entry = tb.Entry(parameters_frame, textvariable=self.parame-
ters_dict["wolves"], width=28)
250.     self.wolves_entry.grid(row=2, column=1, padx=5, pady=5, sticky=EW)
251.
252.     # --- Widget 4: Μέγιστο Πλήθος Επαναλήψεων Αλγορίθμου
253.     tb.Label(parameters_frame, text="Max Iterations:").grid(row=3, column=0,
padx=5, pady=5, sticky=W)
254.     self.parameters_dict["max_iter"] = tk.StringVar(value="100")
255.     self.max_iter_entry = tb.Entry(parameters_frame, textvariable=self.parame-
ters_dict["max_iter"], width=28)
256.     self.max_iter_entry.grid(row=3, column=1, padx=5, pady=5, sticky=EW)
257.
258.     # --- Widget 5: Ελάχιστη Τυπική Απόκλιση
259.     tb.Label(parameters_frame, text="Min Fitness Std Dev:").grid(row=4, col-
umn=0, padx=5, pady=5, sticky=W)
260.     self.parameters_dict["min_std_dev"] = tk.StringVar(value="1e-10")
261.     self.min_std_dev_entry = tb.Entry(parameters_frame, textvariable=self.parame-
ters_dict["min_std_dev"], width=28)
262.     self.min_std_dev_entry.grid(row=4, column=1, padx=5, pady=5, sticky=EW)
263.
264.     # --- Widget 6: Αριθμός Εκτελέσεων
265.     tb.Label(parameters_frame, text="Number of Runs:").grid(row=5, column=0,
padx=5, pady=5, sticky=tb.W)
266.     self.parameters_dict["runs"] = tk.StringVar(value="1")
267.     self.runs_entry = tb.Entry(parameters_frame, textvariable=self.parame-
ters_dict["runs"], width=28)
268.     self.runs_entry.grid(row=5, column=1, padx=5, pady=5, sticky=EW)
269.
270.
271.     def create_plot_control_options_frame(self, parent):
272.
273.         """
274.         Η Συνάρτηση δημιουργεί ένα LabelFrame που περιέχει τα στοιχεία ελέγχου
275.         για τις επιλογές εμφάνισης των γραφημάτων (Ενεργοποίηση Καμπύλης
Σύγκλισης - Χωρικών Γραφημάτων)

```

```

276.         καθώς και τα κουμπιά έναρξης και πάυσης της εκτέλεσης του GWO (RUN-
STOP)
277.
278.         Args:
279.         parent (tk.widget): Περνιέται ως παράμετρος το γονικό widget μέσα στο
οποίο θα δημιουργηθεί
280.         το LabelFrame των Control Widgets.
281.         """
282.
283.         # Δημιουργούμε το LabelFrame και το τοποθετούμε.
284.         plot_controls_lf = tk.LabelFrame(parent, text="Plot's Controls", padding=10,
bootstyle=WARNING)
285.         plot_controls_lf.pack(fill=X, pady=(0, 10))
286.
287.         # --- Widget 1: Toggle για Εμφάνιση Καμπύλης Σύγκλισης Καταλληλότητας
(Convergence Curve).
288.         self.conv_live_update_check = tk.Checkbutton(
289.             plot_controls_lf, # Γονικό LabelFrame
290.             text="Draw Convergence Curve", # Το Κείμενο του Label
291.             variable=self.enable_convergence_plot_var, # Σύνδεση με την Boolean
self.enable_convergence_plot_var
292.             bootstyle="info-round-toggle" # Στύλ Κουμπιού
293.         )
294.         self.conv_live_update_check.grid(row=0, column=0, padx=5, pady=2,
sticky=W)
295.
296.         # --- Widget 2: Toggle για τα Spatial Plots (3D/Top View) ---
297.         self.spatial_live_update_check = tk.Checkbutton(
298.             plot_controls_lf, # Γονέας το LabelFrame
299.             text="Draw Spatial Plots (Only the Active Tab)",
300.             variable=self.enable_spatial_live_update_var,
301.             bootstyle="info-round-toggle"
302.         )
303.         self.spatial_live_update_check.grid(row=1, column=0, padx=5, pady=2,
sticky=W)
304.
305.         # --- Widget 3: Spinbox για Επιλογή Συχνότητας Ενημέρωσης των
Γραφημάτων ---
306.         # Δημιουργούμε ένα Label για την Εμφάνιση Κειμένου σχετικού με την
λειτουργικότητα του SpinBox
307.         freq_label = tk.Label(plot_controls_lf, text="Plot's Update Frequency (plot/iter-
ations):")
308.         freq_label.grid(row=2, column=0, padx=5, pady=5, sticky=W)
309.         # Δημιουργία ανώνυμης μεταβλητής StringVar με αρχική τιμή 5 που ακουεί τις
επιλογές του χρήστη στο spinbox
310.         # και η τιμή καταχωρείται στο parameters_dict["plot_update_freq"]
311.         self.parameters_dict["plot_update_freq"] = tk.StringVar(value="5")
312.         self.plot_update_freq_spinbox = ttk.Spinbox( # Άλλαξε το Entry σε Spinbox
313.             plot_controls_lf, # Γονικό widget

```

```

314.         from_=1, # Σημείο Έναρξης
315.         to=100, # Λήξη
316.         increment=5, # Βήμα
317.         textvariable=self.parameters_dict["plot_update_freq"], # Σύνδεση με
Μεταβλητή Listener
318.         width=5, # πλάτος Spinbox
319.         state="readonly" # Ο χρήστης δεν μπορεί να εισάγει τιμές (ενδεχομένως In-
valid).
320.     )
321.     self.plot_update_freq_spinbox.grid(row=2, column=1, padx=5, pady=5,
sticky=W)
322.
323.     # Δημιουργούμε ένα Frame για να ομαδοποιήσουμε τα κουμπιά
324.     control_frame = tk.Frame(parent)
325.     control_frame.pack(fill=X, pady=5)
326.
327.     # Δημιουργούμε το κουμπί Run για την εκτέλεση του Αλγορίθμου και μόλις
πατιέται καλείται η συνάρτηση start_gwo_thread_wrapper()
328.     self.run_button = tk.Button(control_frame,
329.                                text="Run Algorithm",
330.                                command=self.start_gwo_thread_wrapper,
331.                                bootstyle=SUCCESS,
332.                                width=15)
333.     # Το τοποθετούμε στο αριστερό μέρος του control_frame
334.     self.run_button.pack(side=LEFT, padx=(0,5), expand=True, fill=X)
335.
336.     # Δημιουργούμε κουμπί Stop Algorithm σε περίπτωση που θέλει ο χρήστης να
τερματίσει πρόωρα την εκτέλεση του αλγορίθμου
337.     self.stop_button = tk.Button(control_frame,
338.                                  text="Stop Algorithm",
339.                                  command=self.stop_gwo,
340.                                  bootstyle=DANGER,
341.                                  state=DISABLED,
342.                                  width=15)
343.     # Το τοποθετούμε δίπλα και δεξιά του Run Button
344.     self.stop_button.pack(side=LEFT, padx=5, expand=True, fill=X)
345.
346.     def create_progress_frame(self, parent):
347.         """
348.         Δημιουργούμε ένα Frame που έχει όλα τα πεδία με τις ετικέτες για τα
αποτελέσματα
349.         του αλγορίθμου σε κάθε επανάληψη.
350.
351.         Args:
352.         parent (tk.Widget): Περνιέται ως παράμετρος το γονικό widget μέσα στο
οποίο θα δημιουργηθεί
353.         το LabelFrame των σχετικών με την πρόοδο του αλγορίθμου widgets.
354.         """

```

```

355.     progress_frame = tb.LabelFrame(parent, text="Progress", padding=10, boot-
style=INFO)
356.     progress_frame.pack(fill=X, pady=10)
357.
358.     # Δημιουργούμε τα Labels για τα αποτελέσματα
359.     # --- 1. "Current Run" ---
360.     tb.Label(progress_frame, text="Current Run:").grid(row=0, col-
umn=0, sticky=W, padx=5, pady=2)
361.     self.current_run_label = tb.Label(progress_frame, text="N/A", width=20)
362.     self.current_run_label.grid(row=0, column=1, sticky=W, padx=5, pady=2)
363.
364.     # --- 2. "Current Iteration" ---
365.     tb.Label(progress_frame, text="Current Iteration:").grid(row=1, column=0,
sticky=W, padx=5, pady=2)
366.     self.current_iter_label = tb.Label(progress_frame, text="N/A", width=20)
367.     self.current_iter_label.grid(row=1, column=1, sticky=W, padx=5, pady=2)
368.
369.     # --- 3. "Best Fitness" ---
370.     tb.Label(progress_frame, text="Best Fitness:").grid(row=2, column=0,
sticky=W, padx=5, pady=2)
371.     self.alpha_fitness_label = tb.Label(progress_frame, text="N/A", width=20)
372.     self.alpha_fitness_label.grid(row=2, column=1, sticky=W, padx=5, pady=2)
373.
374.     # --- 4. "Fitness Std Deviation" ---
375.     tb.Label(progress_frame, text="Fitness Std Deviation:").grid(row=3, column=0,
sticky=W, padx=5, pady=2)
376.     self.std_dev_label = tb.Label(progress_frame, text="N/A", width=20)
377.     self.std_dev_label.grid(row=3, column=1, sticky=W, padx=5, pady=2)
378.
379.     # --- 5. "Execution Time" ---
380.     tb.Label(progress_frame, text="Execution Time (sec):").grid(row=4, column=0,
sticky=W, padx=5, pady=2)
381.     self.exec_time_label = tb.Label(progress_frame, text="N/A", width=20)
382.     self.exec_time_label.grid(row=4, column=1, sticky=W, padx=5, pady=2)
383.
384.     ttk.Separator(progress_frame, orient=HORIZONTAL).grid(row=5, column=0,
columnspan=2, sticky=EW, pady=5)
385.
386.     # Δημιουργούμε τη μπάρα προόδου.
387.     self.progress_bar = tb.Progressbar(
388.         progress_frame,
389.         orient=HORIZONTAL,
390.         mode='determinate',
391.         bootstyle=SUCCESS + STRIPED #πράσινο χρώμα με ρίγες.
392.     )
393.     self.progress_bar.grid(row=6, column=0, columnspan=2, sticky=EW,
pady=(10,5))
394.
395.     def create_log_frame(self, parent):

```

```
396.      """
397.      Δημιουργούμε ένα Frame και τοποθετούμε ένα Text widget για την
εμφάνιση πληροφοριών
398.      σχετικών με την εκτέλεση του αλγορίθμου όπως σε ποιο Run είμαστε και
που αποθηκεύθηκε
399.      το αρχείο αποτελεσμάτων.
400.
401.      Args:
402.      parent (tk.widget): Περνιέται ως παράμετρος το γονικό widget μέσα στο
οποίο θα δημιουργηθεί
403.      το LabelFrame των widgets για το Log Frame.
404.      """
405.      log_frame = tk.LabelFrame(parent, text="Log", padding=10, bootstyle=SEC-
ONDARY)
406.      log_frame.pack(fill=BOTH, expand=True, pady=10)
407.
408.      #Δημιουργούμε ένα Text widget για την εμφάνιση πληροφοριακών μηνυμάτων
στον Χρήστη.
409.      self.log_text = tk.Text(log_frame, height=10, wrap=WORD, state=DISA-
BLED)
410.      self.log_text.pack(side=LEFT, fill=BOTH, expand=True)
411.
412.      #Δημιουργούμε μια μπάρα κύλισης (Scrollbar).
413.      scrollbar = ttk.Scrollbar(log_frame, orient=VERTICAL, com-
mand=self.log_text.yview)
414.      scrollbar.pack(side=RIGHT, fill=Y)
415.
416.      # Συνδέουμε το Text widget με το scrollbar.
417.      self.log_text.config(yscrollcommand=scrollbar.set)
418.
419.      def _update_plot_bg_colors(self):
420.          """
421.          private συνάρτηση που ενημερώνει τις μεταβλητές χρώματος φόντου των
γραφημάτων
422.          (self.plot_fig_bg_color, self.plot_axes_bg_color) με βάση την τιμή της
423.          self.plot_dark_background_var.
424.          """
425.          # Αν ο χρήστης ενεργοποιήσει το toggle για σκούρο φόντο στα γραφήματα
426.          # Χρησιμοποιούμε hard coded χρώματα συμβατά και φιλικά στο χρήστη για
μία
427.          # κατα το δυνατόν καλύτερη UX ειδικά όταν γίνεται χρήση dark θέματος για το
gui
428.          if self.plot_dark_background_var.get():
429.              self.plot_fig_bg_color = '#2B3E50'
430.              self.plot_axes_bg_color = '#34495E'
431.          else: # Αλλιώς χρησιμοποιούμε το default λευκό
432.              self.plot_fig_bg_color = 'white'
433.              self.plot_axes_bg_color = 'white'
434.
```

435. #3. Μέθοδοι Δημιουργίας και Αρχικοποίησης Γραφημάτων

```
436.
437. def create_plots_frame(self, parent):
438.     """
439.     Η μέθοδος δημιουργεί το πλαίσιο που περιέχει τα γραφήματα της εφαρμογής
440.     οργανωμένα σε καρτέλες χρησιμοποιώντας το widget ttk.Notebook.
441.
442.     Args:
443.     parent (tk.widget): Περνιέται ως παράμετρος το γονικό widget μέσα στο
444.     οποίο θα δημιουργηθεί το Notebook το οποίο είναι το right_frame.
445.
446.     """
447.
448.     # --- Δημιουργούμε το Notebook widget για να έχουμε καρτέλες.
449.     self.plot_notebook = ttk.Notebook(parent, bootstyle="primary")
450.     self.plot_notebook.pack(fill=BOTH, expand=True, pady=(0, 5)) # Λίγο κενό
κάτω
451.
452.     # Δημιουργούμε ένα λεξικό για να αποθηκεύουμε τα στοιχεία κάθε καρτέλας
453.     self.plot_tabs_dict = {}
454.     # Διατρέχουμε την λίστα με τα λεξικά των ρυθμίσεων κάθε καρτέλας
455.     for config in self.plot_configs_list:
456.         # Αποθηκεύουμε σε κατάλληλες μεταβλητές τα values των αντίστοιχων keys
457.         key = config["key"]
458.         title = config["title"]
459.         projection = config["projection"]
460.
461.         # Δημιουργούμε ένα Frame για να μπει ως καρτέλα του Notebook
462.         tab_frame = tk.Frame(self.plot_notebook, padding=5)
463.         self.plot_notebook.add(tab_frame, text=title) # Προσθέτουμε το frame ως
464.         νέα καρτέλα
465.
466.         # Δημιουργούμε μια Matplotlib Figure που θα περιέχει όλα τα στοιχεία του
467.         γραφήματος
468.         fig = plt.Figure(figsize=(6, 5), dpi=100)
469.
470.         # Προσθέτουμε Άξονες (Axes) στη Figure
471.         if projection == "3d":
472.             ax = fig.add_subplot(111, projection='3d') # πλέγμα 1x1, 3d απεικόνιση
473.         else:
474.             ax = fig.add_subplot(111) # πλέγμα 1x1, 2D απεικόνιση
475.
476.         # Δημιουργούμε τον Canvas του Tkinter που θα φιλοξενήσει τη Figure
477.         canvas = FigureCanvasTkAgg(fig, master=tab_frame)
478.         # Τοποθετούμε τον καμβά μέσα στο tab_frame
479.         canvas.get_tk_widget().pack(side=TOP, fill=BOTH, expand=True)
```

```

478.      # Προσθέτουμε και μία μπάρα εργαλείων του Matplotlib με τυπικά κουμπιά
ενεργειών.
479.      toolbar = NavigationToolbar2Tk(canvas, tab_frame)
480.      toolbar.update()
481.      canvas.get_tk_widget().pack(side=TOP, fill=BOTH, expand=True) #
Επανατοποθέτηση μετά την toolbar
482.
483.      # Αποθηκεύουμε τα στοιχεία της καρτέλας
484.      self.plot_tabs_dict[key] = {"fig": fig, "ax": ax, "canvas": canvas, "toolbar":
toolbar}
485.
486.      # Καλούμε μια μέθοδο για να βάλουμε αρχικούς τίτλους κλπ. στα γραφήματα
487.      self.initialize_plots()
488.
489.      # Επαναφορά (ή αρχικοποίηση) των δεδομένων για την καμπύλη σύγκλισης.
490.      self.convergence_data = {'iterations': [], 'fitness': []}
491.
492.
493.
494.      def initialize_plots(self):
495.          """
496.              Η μέθοδος αρχικοποιεί όλα τα γραφήματα που έχουν δημιουργηθεί και
497.              για κάθε γράφημα προβαίνει στις εξής ενέργειες:
498.              - Καθαρίζει τυχόν προηγούμενα δεδομένα απο την επιφάνεια σχεδίασης,
499.              - Καθορίζει του άξονες με τους τίτλους και τις ετικέτες των αξόνων,
500.              - Εφαρμόζει το τρέχων στυλ της εφαρμογής στα γραφήματα.
501.          """
502.      # Ελέγχουμε αν το λεξικό δεν έχει αρχικοποιηθεί ή είναι κενό τότε
503.      # κάνουμε return.
504.      if not hasattr(self, 'plot_tabs_dict') or not self.plot_tabs_dict:
505.          return
506.
507.      # Αλλιώς διατρέχουμε κάθε ζεύγος K-V του λεξικού self.plot_tabs_dict
508.      for plot_key, elements in self.plot_tabs_dict.items():
509.          ax = elements["ax"]
510.          fig = elements["fig"]
511.          # Με την clear() αφαιρούνται όλα τα στοιχεία που έχουν σχεδιαστεί
512.          # πάνω στους άξονες.
513.          ax.clear()
514.
515.          #--- Ακολουθώς αναλόγως του είδους του γραφήματος γίνεται και αντίστοιχη
αρχικοποίηση γραφήματος.
516.          if plot_key == "3d_view":
517.              ax.set_title('3D View (Benchmark Function with Search Agents) ')
518.              ax.set_xlabel('X1')
519.              ax.set_ylabel('X2')
520.              ax.set_zlabel('Fitness Value')
521.              self.plotted_3d_surface = None # Επιφάνεια Συνάρτησης Καταλληλότητας
522.              self.scatter_3d_wolves = None # Θέσεις Λύκων

```

```

523.         elif plot_key == "top_view":
524.             ax.set_title('Top View (Benchmark Function & Agents)')
525.             ax.set_xlabel('X1')
526.             ax.set_ylabel('X2')
527.             ax.set_aspect('equal', adjustable='box')
528.             if self.plotted_top_view_colorbar:
529.                 try: self.plotted_top_view_colorbar.remove()
530.                 except: pass
531.                 self.plotted_top_view_colorbar = None
532.                 self.plotted_top_view_contour = None
533.                 self.scatter_top_view_wolves = None
534.             elif plot_key == "convergence":
535.                 ax.set_title('Convergence Curve (Best Fitness)')
536.                 ax.set_xlabel('Iteration')
537.                 ax.set_ylabel('Best Fitness')
538.
539.             self._apply_plot_style(ax, fig, plot_key)
540.             elements["canvas"].draw_idle()
541.             # Κάνουμε reset τον canvas αφαιρώντας την καμπύλη και όλα τα σχεδιασμένα
542.             # δεδομένα της καμπύλης σύγκλισης προκειμένου να σχεδιαστεί εκ νέου
543.             # η νέα καμπύλη κατά την εκτέλεση του νέου RUN του GWO.
544.             self.convergence_data = {'iterations': [], 'fitness': []}
545.
546.     def initialize_single_plot_axes(self, plot_key, benchmark_obj):
547.         """
548.         Αρχικοποιεί τους άξονες ενός συγκεκριμένου γραφήματος (που
549.         προσδιορίζεται απο το Plot key),
550.         θέτωντας τους τίτλους και τις ετικέτες του και εφαρμόζοντας το τρέχον
551.         οπτικό στυλ.
552.         Args:
553.         - plot_key(str): Το κλειδί του γραφήματος (πχ top_view) που θα
554.         αρχικοποιηθεί.
555.         - benchmark_obj (BenchmarkFunction): Το αντικείμενο της κλάσης
556.         BenchmarkFunction που αντιστοιχεί
557.         στην τρέχουσα συνάρτηση αξιολόγησης.
558.         """
559.         # Ελέγχουμε αν υπάρχει το λεξικό με τις παραμέτρους του γραφήματος και οτι
560.         # περιλαμβάνεται ως κλειδί
561.         # το plot_key που έχει περάσει ως όρισμα στην συνάρτηση. Αν δεν πληρούνται
562.         # όλες οι συνθήκες επιστρέφουμε.
563.         if not hasattr(self, 'plot_tabs_dict') or plot_key not in self.plot_tabs_dict:
564.             return
565.
566.         # Λαμβάνουμε το configuration για το συγκεκριμένο κλειδί που είναι επίσης
567.         # ένα λεξικό
568.         # και προσπελάνουμε το λεξικό elements και λαμβάνουμε τις τιμές για τα
569.         # κλειδιά ax, fig, canvas

```

```

564.     elements = self.plot_tabs_dict[plot_key]
565.     ax, fig, canvas = elements["ax"], elements["fig"], elements["canvas"]
566.
567.     # Για το 3D θέτουμε τα βασικά στοιχεία των αξόνων
568.     if plot_key == "3d_view":
569.         ax.set_title(f'{benchmark_obj.get_name()} Function - 3D View')
570.         ax.set_xlabel('X1')
571.         ax.set_ylabel('X2')
572.         ax.set_zlabel('Fitness Value')
573.     elif plot_key == "top_view":
574.         ax.set_title(f'{benchmark_obj.get_name()} Function - Top View')
575.         ax.set_xlabel('X1')
576.         ax.set_ylabel('X2')
577.         ax.set_aspect('equal', adjustable='box')
578.         if self.plotted_top_view_colorbar:
579.             self.plotted_top_view_colorbar.set_label("Fitness Value")
580.
581.     self.apply_plot_style(ax, fig, plot_key)
582.     canvas.draw_idle()
583.
584.     def _apply_plot_style(self, ax, fig, plot_key):
585.         """
586.         Εφαρμόζει το τρέχον στυλ (Χρώματα Φόντου, Κειμένου, Grid, Κλπ) σε
587.         κάποιο γράφημα. Την καλούμε στην αρχικοποίηση των γραφημάτων όταν
588.         αλλάζει το θέμα της εφαρμογής ή το φόντο των γραφημάτων απο το toggle
button
589.
590.         Args:
591.         ax (matplotlib.axes.Axes): Αντικείμενο Axes του γραφήματος στο οποίο θα
εφαρμοστεί το στυλ.
592.         fig (matplotlib.figure.Figure): Αντικείμενο Figure του Γραφήματος.
593.         plot_key(str): Το Κλειδί που αντιστοιχεί στο όνομα του γραφήματος (π.χ
top_view)
594.         """
595.         # --- Εφαρμογή Χρωμάτων Φόντου ---
596.         # Εφαρμόζονται τα χρώματα των Figure (γύρω απο την γραφική παράσταση)
597.         # και Axes (άξονες - περιοχή σχεδίασης γραφικής παράστασης).
598.         fig.set_facecolor(self.plot_fig_bg_color)
599.         ax.set_facecolor(self.plot_axes_bg_color)
600.
601.         # --- Καθορισμός Χρωμάτων Κειμένου, Grid και Γραμμών Αξόνων ---
602.         #Λαμβάνουμε το όνομα του τρέχοντος θέματος της εφαρμογής.
603.         current_ttk_theme_name = self.style.theme.name
604.         # Μεταβλητή Boolean αν το θέμα είναι σκούρο
605.         is_ttk_dark = current_ttk_theme_name in self.dark_ttk_themes
606.         # Καθορίζουμε τα χρώματα με βάση το θέμα της εφαρμογής και του φόντου
του γραφήματος.
607.         # Αν ο χρήστης έχει επιλέξει σκούρο φόντο τότε:
608.         if self.plot_dark_background_var.get():

```

```

609.     text_color = 'lightgrey' # Χρώμα κειμένου
610.     title_color = 'white' # Χρώμα Τίτλου
611.     grid_color = '#666666' # Χρώμα Πλέγματος
612.     spine_color = 'lightgrey' # Χρώμα Αξόνων
613.     plot_line_color = 'cyan' if plot_key == "convergence" else 'red' # Χρώμα
Καμπύλης
614.     elif is_ttk_dark: # Αν το θέμα της εφαρμογής είναι σκούρο και το φόντο
ανοιχτό τότε:
615.         text_color = 'lightgrey' # Χρώμα κειμένου
616.         title_color = 'white' # Χρώμα Τίτλου
617.         grid_color = '#555555' # Χρώμα Πλέγματος
618.         spine_color = 'lightgrey' # Χρώμα Αξόνων
619.         plot_line_color = 'cyan' if plot_key == "convergence" else 'red' # Χρώμα
Καμπύλης
620.     else: # Αν το θέμα είναι ανοιχτό και το φόντο είναι ανοιχτόχρωμο τότε
προσαρμόζουμε
621.         # τα χρώματα των αξόνων, του πλέγματος κλπ για βέλτιστη UX:
622.         text_color = '#333333' # Χρώμα κειμένου
623.         title_color = 'black' # Χρώμα Τίτλου
624.         grid_color = 'lightgrey' # Χρώμα Πλέγματος
625.         spine_color = '#333333' # Χρώμα Αξόνων
626.         plot_line_color = 'dodgerblue' if plot_key == "convergence" else 'red' #
Χρώμα Καμπύλης
627.
628.         # --- Εφαρμογή των χρωμάτων στα στοιχεία του γραφήματος ---
629.         # Τίτλος του γραφήματος
630.         ax.title.set_color(title_color)
631.         # Ετικέτες των αξόνων (X,Y)
632.         ax.xaxis.label.set_color(text_color)
633.         ax.yaxis.label.set_color(text_color)
634.         # Αν υπάρχει και άξονας Z τότε καθορίζουμε το χρώμα του:
635.         if hasattr(ax, 'zaxis'):
636.             ax.zaxis.label.set_color(text_color)
637.             ax.tick_params(axis='z', colors=text_color)
638.
639.         ax.tick_params(axis='x', colors=text_color)
640.         ax.tick_params(axis='y', colors=text_color)
641.
642.         for spine in ax.spines.values():
643.             spine.set_edgecolor(spine_color)
644.         # Εμφανίζουμε το πλέγμα και θέτουμε την μορφή του, το χρώμα του και την
πυκνότητά του.
645.         ax.grid(True, linestyle='--', color=grid_color, alpha=0.6)
646.
647.         # Ακολουθώς προβαίνουμε σε επιμέρους μορφοποίηση αναλόγως του είδους
του γραφήματος.
648.         if plot_key == "convergence" and ax.lines:
649.             for line in ax.lines:
650.                 line.set_color(plot_line_color)

```

```

651.
652.     if plot_key == "top_view" and hasattr(self, 'plotted_top_view_colorbar') and
self.plotted_top_view_colorbar:
653.         self.plotted_top_view_colorbar.ax.yaxis.label.set_color(text_color)
654.         self.plotted_top_view_colorbar.ax.tick_params(axis='y', colors=text_color)
655.
656.         for spine in self.plotted_top_view_colorbar.ax.spines.values():
657.             spine.set_edgecolor(spine_color)
658.         # Εφαρμόζουμε tight_layout() για να αποφύγουμε επικάλυψη ετικετών
659.         # Επειδή η tight_layout() προκαλεί εξαιρέσεις τις διαχειριζόμαστε
660.         # τυπώνοντας ένα μήνυμα στο log area για να ενημερωθεί ο χρήστης.
661.         try:
662.             fig.tight_layout(pad=1.5)
663.         except Exception as e:
664.             print(f'Error plotting the graph's: {e}')
665.             self.log_message(f'Error plotting the graph's: {e}')
666.             pass
667.
668.     # 4. Μέθοδοι Ενημέρωσης Περιεχομένου Γραφημάτων
=====
=====
669.
670.     def draw_benchmark_static_backgrounds(self, benchmark_obj):
671.
672.         """
673.         Η συνάρτηση σχεδιάζει την συνάρτηση αξιολόγησης στα γραφήματα 3D
View
674.         και Top View. Καλείται μία φορά όταν αλλάζει η συνάρτηση ή όταν ξεκινά
675.         ο αλγόριθμος για πρώτη φορά.
676.
677.         Args:
678.             benchmark_obj (BenchmarkFunction): Ένα αντικείμενο της κλάσης
BenchmarkFunction που αποτελεί
679.             και την συνάρτηση αξιολόγησης η οποία θα σχεδιαστεί.
680.         """
681.
682.         # Ελέγχουμε αν υπάρχει το λεξικό με τις ρυθμίσεις σχεδίασης των γραφημάτων
683.         # Αν δεν υπάρχουν τότε απλά γίνεται return απο την συνάρτηση
684.         if not hasattr(self, "plot_tabs_dict") or not self.plot_tabs_dict:
685.             print("DEBUG: Plot configurations dictionary doesn't exists.")
686.             return
687.
688.         # Έλεγχος αν έχουμε πάνω απο δύο διαστάσεις του προβλήματος και άρα μαζί
με την διάσταση
689.         # που αντιστοιχεί στις τιμές καταλληλότητας πάνω απο 3 διαστάσεις. Σε αυτή
την περίπτωση
690.         # δεν μπορούμε να σχεδιάσουμε τα χωρικά γραφήματα και καλούμε την clear()
για καθαρισμό των
691.         # σχεδιάσεων απο προηγούμενες εκτελέσεις.

```

```

692.     if benchmark_obj.dimensions != 2:
693.         if "3d_view" in self.plot_tabs_dict:
694.             elements = self.plot_tabs_dict["3d_view"]
695.             ax_3d, canvas_3d = elements["ax"], elements["canvas"]
696.             ax_3d.clear()
697.             self.initialize_single_plot_axes("3d_view", benchmark_obj)
698.             self.plotted_3d_surface = None
699.             canvas_3d.draw_idle()
700.         if "top_view" in self.plot_tabs_dict:
701.             elements = self.plot_tabs_dict["top_view"]
702.             ax_top, canvas_top = elements["ax"], elements["canvas"]
703.             ax_top.clear()
704.             if self.plotted_top_view_colorbar:
705.                 try:
706.                     self.plotted_top_view_colorbar.remove()
707.                 except:
708.                     pass
709.             self.plotted_top_view_colorbar = None
710.             self.initialize_single_plot_axes("top_view", benchmark_obj)
711.             self.plotted_top_view_contour = None
712.             canvas_top.draw_idle()
713.
714.             self.current_benchmark_object_for_spatial_plot = None
715.             return # Σταματάμε εδώ, αφού δεν θα γίνει σχεδίαση
716.             # Σε άλλη περίπτωση, δηλαδή έχουμε 3 διαστάσεις, λαμβάνουμε το αντικείμενο
717.             # της συνάρτησης αξιολόγησης και το αποθηκεύουμε στο πεδίο self.cur-
718.             # rent_benchmark_object_for_spatial_plot.
719.             #print(f'DEBUG: Drawing static background for {benchmark_obj.name}')
720.             self.current_benchmark_object_for_spatial_plot = benchmark_obj
721.             # Ακολουθώς παίρνουμε τα όρια της συνάρτησης και τα αποθηκεύουμε σε
722.             # αντίστοιχες μεταβλητές.
723.             lower_bound, upper_bound = benchmark_obj.lower_bound, benchmark_obj.up-
724.             per_bound
725.             # Καθορίζουμε ένα πλήθος από σημεία για την δημιουργία του πλέγματος.
726.             num_points = 75
727.             # Δημιουργούνται δύο πίνακες με num_points πλήθος σημείων ομοιόμορφα
728.             # κατανεμημένα ανάμεσα στα όρια.
729.             X1_vals = np.linspace(lower_bound, upper_bound, num_points) # []
730.             X2_vals = np.linspace(lower_bound, upper_bound, num_points) # []
731.             # Δημιουργούνται δύο δισδιάστατοι πίνακες με τις συνταταγμένες X, Y των
732.             # σημείων του πλέγματος.
733.             X, Y = np.meshgrid(X1_vals, X2_vals)
734.             # Υπολογίζουμε τις τιμές για τον άξονα Z που είναι οι τιμές fitness για κάθε
735.             # σημείο (x,y) του πλέγματος.
736.             # Αρχικά εκτελούμε αρχικοποίηση του πίνακα Z με μηδενικά, ίδιων
737.             # διαστάσεων με το X (ή το Y).
738.             Z = np.zeros_like(X)

```

```

734.     for i in range(X.shape[0]): # Για κάθε σειρά του πλέγματος
735.         for j in range(X.shape[1]): # Για κάθε στήλη του πλέγματος
736.             # καλείται η συνάρτηση υπολογισμού της καταλληλότητας calculate_fit-
ness() για
737.             # τις τρέχουσες συντεταγμένες του πλέγματος και υπολογίζεται το fitness.
738.             Z[i, j] = benchmark_obj.calculate_fitness([X[i, j], Y[i, j]])
739.
740.     # Σχεδιάζουμε το Background του 3D View Γραφήματος της συνάρτησης
αξιολόγησης.
741.     if "3d_view" in self.plot_tabs_dict:
742.         ax_3d = self.plot_tabs_dict["3d_view"]["ax"]
743.         canvas_3d = self.plot_tabs_dict["3d_view"]["canvas"]
744.         ax_3d.clear() # Καθαρισμός των αξόνων πριν την σχεδίαση
745.         # Σχεδίαση με την plot_surface() της συνάρτησης αξιολόγησης με
συγκεκριμένες παραμέτρους
746.         # (π.χ χρωματική παλέτα viridis, μπορούμε να διαλέξουμε και inferno)
747.         self.plotted_3d_surface = ax_3d.plot_surface(X, Y, Z, cmap="viridis", al-
pha=0.7, edgecolor="none")
748.         #ax_3d.legend(loc='upper right', fontsize='small', framealpha=0.5)
749.         self.initialize_single_plot_axes("3d_view", benchmark_obj)
750.         canvas_3d.draw_idle()
751.     # Ομοίως για το Top View
752.     if "top_view" in self.plot_tabs_dict:
753.         ax_top = self.plot_tabs_dict["top_view"]["ax"]
754.         canvas_top = self.plot_tabs_dict["top_view"]["canvas"]
755.         fig_top = self.plot_tabs_dict["top_view"]["fig"]
756.         ax_top.clear() # Καθαρισμός των αξόνων πριν την σχεδίαση
757.         if self.plotted_top_view_contour: # Αφαίρεση παλαιού colorbar αν υπάρχει
758.             try:
759.                 self.plotted_top_view_colorbar.remove()
760.             except Exception as e:
761.                 #print(f'DEBUG: Error removing old colorbar: {e}')
762.                 self.plotted_top_view_colorbar = None
763.
764.         self.plotted_top_view_contour = ax_top.contourf(X, Y, Z, levels=50,
cmap='viridis', alpha=0.8)
765.         self.plotted_top_view_colorbar = fig_top.colorbar(self.plot-
ted_top_view_contour, ax=ax_top, orientation='vertical', shrink=0.8)
766.         self.initialize_single_plot_axes("top_view", benchmark_obj)
767.         canvas_top.draw_idle()
768.
769.
770.     def update_convergence_plot(self, clear_only=False):
771.         """
772.         Ενημερώνει ή καθαρίζει το γράφημα της καμπύλης σύγκλισης.
773.
774.         Args:
775.             clear_only(bool): Αν είναι True τότε γίνεται μόνο καθαρισμός του
γραφήματος χωρίς να σχεδιαστούν νέα δεδομένα.
776.

```

```

777.     """
778.     if not hasattr(self, "plot_tabs_dict") or "convergence" not in self.plot_tabs_dict:
779.         return
780.     # Προσπελάζουμε τα δεδομένα σχεδίασης του γραφήματος
781.     elements = self.plot_tabs_dict["convergence"]
782.     ax, fig, canvas = elements["ax"], elements["fig"], elements["canvas"]
783.     # Καθαρίζουμε προηγούμενη σχεδίαση.
784.     ax.clear()
785.
786.     if not clear_only and hasattr(self, 'convergence_data') and self.conver-
gence_data['iterations'] and \
787.         self.convergence_data['fitness']:
788.         if len(self.convergence_data['iterations']) == len(self.convergence_data['fit-
ness']):
789.             ax.plot(self.convergence_data['iterations'], self.convergence_data['fitness'],
790.                     marker='.', linestyle='-', markersize=4)
791.
792.
793.             ax.set_title('Convergence Curve')
794.             ax.set_xlabel('Iteration')
795.             ax.set_ylabel('Best Fitness')
796.             self.apply_plot_style(ax, fig, "convergence")
797.             canvas.draw_idle()
798.
799.     def update_3d_plot_content(self, msg_data):
800.         """
801.         Η συνάρτηση καλείται στο process_gui_queue() όταν λαμβάνεται
μύνημα τύπου "spatial_plot_data"
802.         και η ενεργή καρτέλα στο notebook είναι η "3D_View". Ενημερώνει το
περιεχόμενο του 3D γραφήματος
803.         (καρτέλα 3D View) με τις τρέχουσες θέσεις των πρακτόρων αναζήτησης
(λύκων).
804.
805.         Args: msg_data(dict): Λεξικό με τα δεδομένα για την ενημέρωση του
γραφήματος.
806.         """
807.         if "3d_view" not in self.plot_tabs_dict: return
808.         elements = self.plot_tabs_dict["3d_view"]
809.         ax, canvas = elements["ax"], elements["canvas"]
810.
811.         positions = msg_data.get("wolves_positions")
812.         fitness_values = msg_data.get("wolves_fitness")
813.
814.         if positions is None or fitness_values is None or positions.shape[1] < 2:
815.             return
816.
817.         if positions.shape[0] != fitness_values.shape[0]:
818.             return
819.

```

```

820.     # Αφαίρεσε του λύκους απο το γράφημα
821.     if self.scatter_3d_wolves:
822.         try:
823.             self.scatter_3d_wolves.remove()
824.         except:
825.             pass
826.         self.scatter_3d_wolves = None
827.
828.     # Σχεδιάσε του λύκους στο γράφημα με βάση τις συντεταγμένες των θέσεων
829.     x_coords, y_coords = positions[:, 0], positions[:, 1]
830.     z_coords = fitness_values
831.     self.scatter_3d_wolves = ax.scatter(x_coords, y_coords, z_coords, color='red',
832. s=30, depthshade=True, label="Wolves")
833.     canvas.draw_idle()
834.
835.     def update_top_view_plot_content(self, msg_data):
836.         """
837.         Ομοίως με την update_3d_plot_content(), ενημερώνει το γράφημα με τις
838.         θέσεις των λύκων.
839.         Args: msg_data(dict): Λεξικό με τα δεδομένα για την ενημέρωση του
840.         γραφήματος.
841.         """
842.         if "top_view" not in self.plot_tabs_dict: return
843.         elements = self.plot_tabs_dict["top_view"]
844.         ax, canvas = elements["ax"], elements["canvas"]
845.         positions = msg_data.get("wolves_positions")
846.         if positions is None or positions.shape[1] < 2:
847.             return
848.
849.         # Αφαίρεσε του λύκους απο το γράφημα
850.         if self.scatter_top_view_wolves:
851.             try:
852.                 self.scatter_top_view_wolves.remove()
853.             except:
854.                 pass
855.             self.scatter_top_view_wolves = None
856.
857.         self.scatter_top_view_wolves = ax.scatter(positions[:, 0], positions[:, 1],
858. color='red', s=30, label="Wolves")
859.         canvas.draw_idle()
860.
861.     # 5. Μέθοδοι Χειρισμού Γεγονότων (Events)

```

```
862.
863. def on_benchmark_select(self, event=None):
864.     """
865.     Καλείται όταν ο χρήστης επιλέξει στο combobox κάποια συνάρτηση
αξιολόγησης
866.     από αυτές που έχουμε στην λίστα με τις συναρτήσεις σταθερών διαστάσεων
τότε το
867.     πεδίο εισαγωγής διαστάσεων απενεργοποιείται και τοποθετείται αυτόματα η
τιμή 2
868.     αλλιώς το πεδίο παραμένει ενεργό για εισαγωγή της εισόδου του χρήστη.
869.     """
870.     selected_function = self.parameters_dict["benchmark"].get()
871.     if selected_function in self.fixed_dim_functions:
872.         self.parameters_dict["dimensions"].set("2")
873.         self.dimensions_entry.config(state="disabled")
874.     else:
875.         self.dimensions_entry.config(state="normal")
876.
877. def on_theme_select(self, event=None, update_plots=True):
878.     """
879.     Η Συνάρτηση αυτή λαμβάνει το θέμα που επέλεξε ο χρήστης από το
880.     combobox επιλογής θεμάτων και το ttkbootstrap το εφαρμόζει σε όλη την ε-
φαρμογή
881.     """
882.     selected_theme_name = self.theme_combo.get()
883.     self.style.theme_use(selected_theme_name)
884.     if update_plots and hasattr(self, 'plot_tabs_dict') and self.plot_tabs_dict:
885.         # Εφαρμογή του στυλ του θέματος της εφαρμογής στα γραφήματα.
886.         for plot_key in self.plot_tabs_dict:
887.             elements = self.plot_tabs_dict[plot_key]
888.             self._apply_plot_style(elements["ax"], elements["fig"], plot_key)
889.             elements["canvas"].draw_idle()
890.
891. def on_plot_background_toggle(self):
892.     """
893.     Η μέθοδος καλείται όταν ο Χρήστης επιλέξει μία από τις 2 καταστάσεις
894.     του toggle button για την επιλογή dark/light φόντου γραφικών παραστάσεων.
895.     """
896.     # Αρχικά καλείται η private μέθοδος _update_plot_bg_colors() η οποία
διαβάζει
897.     # την τρέχουσα τιμή του πεδίου self.plot_dark_background_var και ορίζει τα
πεδία
898.     # self.plot_fig_bg_color και self.plot_axes_bg_color.
899.     self._update_plot_bg_colors()
900.     # Εμφανίζουμε μήνυμα στο Log frame για την ενημέρωση του Χρήστη ότι
άλλαξε το φόντο.
901.     self.log_message(f'{'Dark' if self.plot_dark_background_var.get() else 'Light'}
plot background selected.")
```

```

902.      # Έλεγχος ότι υπάρχει και ότι δεν είναι κενό το λεξικό που περιέχει τις
αναφορές στα στοιχεία
903.      # (figure, axes, canvas) κάθε καρτέλας του γραφήματος.
904.      if hasattr(self, 'plot_tabs_dict') and self.plot_tabs_dict:
905.
906.          current_app_theme_is_dark = self.style.theme.name in self.dark_ttk_themes
907.          # Για κάθε κλειδί του λεξικού με τις ρυθμίσεις των καρτελών του Notebook:
908.          for plot_key in self.plot_tabs_dict:
909.              # Παίρνουμε το Value που είναι ένα λεξικό με κλειδιά fig, ax, canvas
910.              elements = self.plot_tabs_dict[plot_key]
911.              # Καλούμε την συνάρτηση _apply_plot_style() περνώντας ως όρισμα τα
values των keys "ax", "fig"
912.              # καθώς και το key των ρυθμίσεων (πχ convergence ή 3d_view ή
top_view)
913.              self._apply_plot_style(elements["ax"], elements["fig"], plot_key)
914.              # Ενημέρωση του canvas για την επανασχεδίαση του γραφήματος με το
νέο στυλ (φόντο, συμβατά χρώματα αξόνων κλπ)
915.              elements["canvas"].draw_idle()
916.
917.      # 6. Μέθοδοι Σχετικές με την Εκτέλεση του GWO και την Επικοινωνία μεταξύ των
Νημάτων =====
918.
919.      def start_gwo_thread_wrapper(self):
920.          """
921.              Αυτή η συνάρτηση εκτελεί τον αλγόριθμο σε ένα νέο νήμα, διαβάζει και
επικυρώνει τις τιμές
922.              από την φόρμα εισόδου των παραμέτρων και ενημερώνει την κατάσταση των
κουμπιών.
923.          """
924.          # Ελέγχουμε αν υπάρχει ήδη το thread για την εκτέλεση του GWO και τρέχει.
925.          if self.gwo_thread and self.gwo_thread.is_alive():
926.              messagebox.showwarning("The GWO is already running.")
927.              return
928.          # Συλλέγουμε και επικυρώνουμε τις τιμές των παραμέτρων από την φόρμα.
929.          try:
930.              # Λεξικό για την αποθήκευση των παραμέτρων.
931.              temp_parameters = {}
932.              self.plot_update_counter = 0
933.
934.              # Διατρέχουμε το λεξικό self.parameters_dict που περιέχει τις μεταβλητές
StringVar από τα πεδία της φόρμας.
935.              for key, value in self.parameters_dict.items():
936.                  # Για κάθε ζεύγος K-V του λεξικού parameters_dict αποθηκεύουμε το ζεύγος
στο λεξικό parameters.
937.                  temp_parameters[key]=value.get()
938.
939.              #Ακολουθώς εκτελούμε επικύρωση και μετατροπή τύπων των τιμών των
παραμέτρων
940.              benchmark_name = str(temp_parameters["benchmark"])

```

```
941.      # Αν η συνάρτηση αξιολόγησης δεν είναι στην λίστα με τις συναρτήσεις
αξιολόγησης
942.      # ή έχει τιμή το Placeholder του πεδίου ενημερώνουμε τον χρήστη για το
λάθος.
943.      if not benchmark_name or benchmark_name == "Select a function...":
944.          messagebox.showerror("Invalid Input", "Please select a valid benchmark
function")
945.          return
946.
947.      # Μετατρέπουμε ακολούθως τους τύπους των τιμών των παραμέτρων απο
string σε σωστούς τύπους (int, float, boolean).
948.      dimensions = int(temp_parameters["dimensions"])
949.      wolves_num = int(temp_parameters["wolves"])
950.      max_iter = int(temp_parameters["max_iter"])
951.      min_std_dev = float(temp_parameters["min_std_dev"])
952.      num_runs = int(temp_parameters["runs"])
953.      plot_enabled_for_gwo = self.enable_spatial_live_update_var.get()
954.
955.      # Ελέγχουμε αν ο χρήστης έχει επιλέξει τον ελάχιστο αριθμό των 3 λύκων
956.      # που απαιτείται για να εκτελεστεί ο αλγόριθμος.
957.      if wolves_num < 3:
958.          messagebox.showerror("Invalid Input", "Choose at least 3 wolves")
959.          return
960.      # Σε περίπτωση επιλογής απο τον χρήστη συνάρτησης σταθερών διαστάσεων
τοποθετούμε στην μεταβλητή
961.      # dimensions την τιμή 2 σε συμφωνία με την ανάθεση στο κατάλληλο widget
να εμφανίσει την τιμή 2.
962.      if benchmark_name in self.fixed_dim_functions:
963.          dimensions = 2
964.
965.      if max_iter <= 0:
966.          messagebox.showerror("Invalid Input", "Max Iterations must be a positive
integer.")
967.          return
968.
969.      if num_runs <= 0:
970.          messagebox.showerror("Invalid Input", "Number of runs must be a posi-
tive integer.")
971.
972.      except ValueError as ex:
973.          messagebox.showerror("Invalid Input", f"error in parameter value: {ex}.\n
Check the input fields")
974.          return
975.
976.      # Προετοιμασία GUI
977.      self.run_button.config(state=DISABLED) # Απενεργοποίηση του κουμπιού
RUN.
978.      self.stop_button.config(state=NORMAL) # Ενεργοποίηση του κουμπιού STOP.
```

```
979.      # Κάνουμε reset το event για να επιτραπεί η εκτέλεση του αλγορίθμου (δεν έχει
πατηθεί STOP).
980.      self.stop_event.clear()
981.
982.      self.cumulative_execution_time = 0.0
983.      self.exec_time_label.config(text="0.0000")
984.      # Καθάρισε τα δεδομένα της καμπύλης σύγκλισης για τη νέα εκτέλεση
985.      self.convergence_data = {'iterations': [], 'fitness': []}
986.      self.update_convergence_plot(clear_only=True) # Καθάρισε το γράφημα
987.
988.      current_benchmark_name = getattr(self.current_benchmark_object_for_spa-
tial_plot, 'name', None)
989.      current_benchmark_dims = getattr(self.current_benchmark_object_for_spa-
tial_plot, 'dimensions', None)
990.
991.      if benchmark_name != current_benchmark_name or \
992.         dimensions != current_benchmark_dims or \
993.         self.current_benchmark_object_for_spatial_plot is None:
994.
995.         plot_benchmark_obj = BenchmarkFunction(benchmark_name, dimensions)
996.         self.draw_benchmark_static_backgrounds(plot_benchmark_obj)
997.         elif dimensions != 2 and self.current_benchmark_object_for_spatial_plot is not
None:
998.         plot_benchmark_obj = BenchmarkFunction(benchmark_name, dimensions) #
Will trigger clearing
999.         self.draw_benchmark_static_backgrounds(plot_benchmark_obj)
1000.
1001.
1002.         self.log_message(f"--- Starting GWO for {benchmark_name} function ---")
1003.
1004.         # Δημιουργία και Εκκίνηση του Thread. target: Η συνάρτηση που θα
εκτελεστεί μέσα στο thread.
1005.         # args: Μια πλειάδα (tuple) με τα ορίσματα που θα περάσουμε στη συνάρτηση
target.
1006.         self.gwo_thread = threading.Thread(
1007.             target = self.run_gwo_algorithm,
1008.             args=(benchmark_name, dimensions, wolves_num, max_iter, min_std_dev,
num_runs, plot_enabled_for_gwo)
1009.         )
1010.         # Αν κλείσει η διεργασία του GUI θα τερματιστεί και το thread.
1011.         self.gwo_thread.daemon = True
1012.         # Εκκίνηση του thread.
1013.         self.gwo_thread.start()
1014.
1015.
1016.         def run_gwo_algorithm(self, benchmark_name, dimensions, wolves_num,
max_iter, min_std_dev, num_runs, plot_enabled):
1017.             """
```

```

1018.      Εκτελεί τον αλγόριθμο GWO σε ένα ξεχωριστό thread για να μην παγώνει το
GUI.
1019.      Στέλνει ενημερώσεις κατάστασης και αποτελέσματα πίσω στο κυρίως thread
του GUI μέσω μίας thread-safe queue.
1020.
1021.      Args:
1022.          benchmark_name (str): Συνάρτηση αξιολόγησης.
1023.          dimensions (int): Πλήθος διαστάσεων του προβλήματος.
1024.          wolves_num (int): Πλήθος πρακτόρων (λύκων).
1025.          max_iter (int): Μέγιστος αριθμός επαναλήψεων.
1026.          min_std_dev (float): Κατώφλι τυπικής απόκλισης.
1027.          num_runs (int): Συνολικός αριθμός εκτελέσεων που θα γίνουν.
1028.          plot_enabled (bool): Επιλογή σχεδίασης γραφικών παραστάσεων.
1029.      """
1030.      final_results_for_gui = None
1031.      try:
1032.          # Δημιουργούμε το directory αν δεν υπάρχει ήδη στο οποίο θα αποθηκευτεί
το csv αρχείο με τα αποτελέσματα.
1033.          results_folder = f'gui_results/results_with_{wolves_num}_agents_gui'
1034.          results_folder = os.path.join("gui_results", f're-
sults_with_{wolves_num}_agents_gui')
1035.          os.makedirs(results_folder, exist_ok=True)
1036.          # Λίστα για την αποθήκευση των αποτελεσμάτων σε κάθε run του
αλγορίθμου.
1037.          results_list_for_csv = []
1038.
1039.          # Επανάληψη για την εκτέλεση του αλγορίθμου για το πλήθος των run που
έχει ορίσει ο χρήστης.
1040.          for i in range(num_runs):
1041.              # Ελεγχος αν ο χρήστης έχει πατήσει το κουμπί STOP.
1042.              if self.stop_event.is_set():
1043.                  # Αν ναι τότε στέλνουμε ένα μήνυμα στο log ότι ο χρήστης ακύρωσε
την εκτέλεση.
1044.                  self.gui_queue.put(("log", "Run aborted by user.))
1045.                  break
1046.
1047.                  # Αποστολή μηνύματος στην ουρά για την εκκαθάριση της καμπύλης
σύγκλισης.
1048.                  self.gui_queue.put(("clear_convergence_for_new_run", {"current_run": i
+ 1, "total_runs": num_runs}))
1049.
1050.                  # Αλλιώς στέλνουμε στο GUI ενημέρωση για τον αριθμό του τρέχοντος
RUN.
1051.                  self.gui_queue.put(("run_update", (i + 1, num_runs)))
1052.                  # Δημιουργούμε ένα αντικείμενο της κλάσης BenchmarkFunction.
1053.                  benchmark_func = BenchmarkFunction(benchmark_name, dimensions)
1054.
1055.                  # Καλούμε την συνάρτηση GWO(), εκτελούμε τον αλγόριθμο με τις
παραμέτρους του χρήστη και αποθηκεύουμε τα αποτελέσματα

```

```
1056.      # σε ένα λεξικό run_results.
1057.      run_results = GWO(
1058.          wolves_num,
1059.          max_iter,
1060.          min_std_dev,
1061.          benchmark_func,
1062.          plot_enabled,
1063.          # Συνάρτηση που ενημερώνει ασύγχρονα το νήμα του GUI απο το
τρέχων νήμα του αλγορίθμου.
1064.          gui_callback=self.gui_callback_from_thread,
1065.          # Αντικείμενο threading.Event για εξωτερική διακοπή (Κουμπί STOP).
1066.          stop_event=self.stop_event
1067.      )
1068.
1069.      if "error" in run_results:
1070.          self.gui_queue.put(("log", f"GWO Error: {run_results['error']}"))
1071.          break
1072.
1073.      run_results['run'] = i + 1 # Προσθέτουμε +1 στο RUN γιατί ξεκινάει απο
το 0.
1074.      # Προσθέτουμε το λεξικό που επέστρεψε το GWO() σαν στοιχείο στο
τέλος της λίστας π.χ [{},{}] με τα αποτελέσματα.
1075.      results_list_for_csv.append(run_results)
1076.      # Κρατάμε ένα αντίγραφο των αποτελεσμάτων για χρήση στο GUI
1077.      final_results_for_gui = run_results.copy()
1078.
1079.      # Παίρνουμε τον χρόνο εκτέλεσης απο τα αποτελέσματα του
1080.      # αλγορίθμου και τον στέλνουμε στην ουρά για κατανάλωση απο το
thread του GUI.
1081.      current_run_time = run_results.get("gwo_time", 0.0)
1082.      self.gui_queue.put(("run_completed_stats", {"gwo_time": cur-
rent_run_time}))
1083.
1084.      # Παίρνουμε το value του Key min_fitness, αν δεν υπάρχει στο λεξικό
τοτε παίρνουμε το N/A.
1085.      fitness = run_results.get('min_fitness', 'N/A')
1086.      # Χρησιμοποιούμε try-except για την περίπτωση που το fitness δεν είναι
αριθμός.
1087.      try:
1088.          # φτιάχνουμε ενα string με την τιμή του fitness σε επιστημονική μορφή
με 4 δεκαδικά π.χ 1.0787E-10
1089.          # Αν για κάποιο λόγο το fitness δεν έχει σωστό τύπο (π.χ είναι το N/A)
ή σωστή τιμή
1090.          # θα πεταχτεί εξαίρεση και θα την πιάσουμε.
1091.          log_msg = f"Run {i+1} finished. Min Fitness: {fitness:.4e}"
1092.      except (TypeError, ValueError):
1093.          # Σε περίπτωση εξαίρεσης προσθέτουμε στο string την υπάρχουσα τιμή
fitness όπως είναι π.χ N/A.
1094.          log_msg = f"Run {i+1} finished. Min Fitness: {fitness}"
```

```
1095.         # Ακολουθώς στέλνουμε στο Thread του GUI μέσω της ουράς το
1096.         # ενημερωτικό μήνυμα με το tag "log".
1097.         self.gui_queue.put(("log", log_msg))
1098.         #Αν η λίστα με τα αποτελέσματα δεν είναι κενή και ο χρήστης δεν έχει
1099.         # πατήσει το κουμπί STOP.
1100.         if results_list_for_csv and not self.stop_event.is_set():
1101.             # Δημιουργούμε το αρχείο csv για τα αποτελέσματα.
1102.             csv_filename = os.path.join(results_folder, f'gwo_results_{bench-
1103.             mark_name}.csv')
1104.             # Παίρνουμε τα ονόματα των πεδίων.
1105.             fieldnames = list(results_list_for_csv[0].keys())
1106.             # Ανοίγουμε το αρχείο για εγγραφή.
1107.             with open(csv_filename, mode="w", newline="", encoding='utf-8') as file:
1108.                 # Δημιουργούμε ένα αντικείμενο csv.DictWriter.
1109.                 writer = csv.DictWriter(file, fieldnames=fieldnames, delimiter=';', ex-
1110.                 trasaction='ignore')
1111.                 # Γράφουμε στο αρχείο την πρώτη γραμμή με τις επικεφαλίδες.
1112.                 writer.writeheader()
1113.                 # Εν συνέχεια γράφουμε και τα αποτελέσματα από την λίστα των
1114.                 # αποτελεσμάτων.
1115.                 writer.writerows(results_list_for_csv)
1116.                 # Στέλνουμε ένα μήνυμα στο GUI μέσω της ουράς για την επιτυχή
1117.                 # αποθήκευση των αποτελεσμάτων στο αρχείο.
1118.                 self.gui_queue.put(("log", f"Results saved to {csv_filename}"))
1119.                 # Αν προκληθεί οποιαδήποτε εξαίρεση την πιάνουμε και στέλνουμε μήνυμα
1120.                 # στο GUI.
1121.                 except Exception as e:
1122.                     self.gui_queue.put(("log", f"An unexpected error occurred in GWO thread:
1123.                     {e}"))
1124.                 # Σε κάθε περίπτωση στέλνουμε μήνυμα ολοκλήρωσης της εκτέλεσης στο GUI
1125.                 # μέσω της ουράς.
1126.                 finally:
1127.                     self.gui_queue.put(("all_runs_finished", final_results_for_gui))
1128.
1129.
1130.
1131. def process_gui_queue(self):
1132.     """
1133.     Η συνάρτηση διαχειρίζεται τα μηνύματα από την thread safe Ουρά που
1134.     χρησιμοποιούμε για την επικοινωνία
1135.     μεταξύ των νημάτων του GWO και του GUI. Καλείται περιοδικά από το
1136.     tkinter event loop και διαβάζει όλα
1137.     τα διαθέσιμα μηνύματα από την ουρά και ενημερώνει τα αντίστοιχα GUI
1138.     Widgets.
1139.     """
1140.     try:
1141.         for _ in range(10): # Κατανάλωσε έως 10 μηνύματα από την ουρά
```

```
1132.         msg_type = None # Αρχικοποίηση τύπου μηνύματος
1133.         msg_data = None # Αρχικοποίηση δεδομένων μηνύματος
1134.         message_successfully_retrieved = False
1135.
1136.         try:
1137.             # Λήψη μηνύματος από την ουρά χωρίς να μπλοκάρει το νήμα αν η
Ουρά είναι άδεια
1138.             # προκαλείται queue.Empty Exception.
1139.             message_item = self.gui_queue.get_nowait()
1140.             # Έλεγχος αν το message_item είναι το αναμενόμενο tuple (τύπος,
δεδομένα).
1141.             if isinstance(message_item, tuple) and len(message_item) == 2:
1142.                 #Αποθηκεύουμε τα στοιχεία του tuple σε μεταβλητές
1143.                 msg_type, msg_data = message_item
1144.                 # Ενημέρωνουμε μία σημαία για επιτυχή λήψη μηνύματος
1145.                 message_successfully_retrieved = True
1146.             else:
1147.                 # Αλλιώς τυπώνουμε στην κονσόλα για σκοπούς debugging.
1148.                 error_log_msg = f"Error with message: {message_item}"
1149.                 print(error_log_msg)
1150.                 # Και το εμφανίζουμε και στο log widget για ενημέρωση του
Χρήστη.
1151.                 self.log_message(error_log_msg)
1152.                 # Σε περίπτωση που η ουρά είναι και προκλαίεται εξαίρεση, πιάνουμε την
εξαίρεση και σπάμε το Loop.
1153.                 except queue.Empty:
1154.                     break
1155.                 # Σε περίπτωση έγερσης άλλης εξαίρεσης την τυπώνουμε στην κονσόλα.
1156.                 except Exception as e_get:
1157.                     error_log_msg_get = f"Error: {e_get}"
1158.                     print(error_log_msg_get)
1159.                     # και εμφανίζουμε και το error στο Log widget του GUI.
1160.                     self.log_message(error_log_msg_get)
1161.                     break # Τερματίζουμε το Loop.
1162.                 # Αν η σημαία επιτυχούς λήψης είναι True και ο τύπος μηνύματος δεν
είναι None τότε:
1163.                 if message_successfully_retrieved and msg_type is not None:
1164.                     # Αν είναι μήνυμα τύπου "log".
1165.                     if msg_type == "log":
1166.                         # Εμφανίζουμε στο Log Widget του GUI το Μήνυμα για να
ενημερωθεί ο Χρήστης.
1167.                         self.log_message(msg_data)
1168.
1169.                         # Αν είναι μήνυμα τύπου "clear_convergence_for_new_run" το οποίο
1170.                         # στέλνεται απο το νήμα του GWO στην αρχή κάθε RUN για να γίνει
clear()
1171.                         # στην καμπύλη σύγκλισης απο τα παλιά δεδομένα.
1172.                         elif msg_type == "clear_convergence_for_new_run":
```

```
1173.          # Ελέγχουμε αν είναι ενεργοποιημένο το Toggle Button για την
εμφάνιση της καμπύλης σύγκλισης καταλληλότητας
1174.          if self.enable_convergence_plot_var.get():
1175.          # Εμφανίζουμε στο Log Widget μήνυμα στο Χρήστη για την
εκκαθάριση του προηγούμενου γραφήματος
1176.          self.log_message(f"Clearing convergence data for new run:
{msg_data['current_run']}/{msg_data['total_runs']}")
1177.          # Μηδενίζουμε τα δεδομένα του γραφήματος της σύγκλισης με
την εκ νέου αρχικοποίηση της μεταβλητής convergence_data
1178.          self.convergence_data = {'iterations': [], 'fitness': []}
1179.          # Κλήση της μεθόδου update_convergence_plot() για την
εκτέλεση του clear().
1180.          self.update_convergence_plot(clear_only=True)
1181.
1182.          # Αν είναι μήνυμα τύπου "run_update", που ενημερώνει το GUI για την
έναρξη του
1183.          # νέου RUN, τότε:
1184.          elif msg_type == "run_update":
1185.          # Ελέγχουμε ομοίως με πριν τον τύπο δεδομένων του μηνύματος (tu-
ple) και το μέγεθος του
1186.          if isinstance(msg_data, tuple) and len(msg_data) == 2:
1187.          # Εξάγουμε τα δεδομένα του tuple
1188.          current_run, total_runs = msg_data
1189.          # Ενημερώνουμε το αντίστοιχο Label με την τιμή του τρέχοντος
RUN για να το δει ο Χρήστης.
1190.          self.current_run_label.config(text=f"{current_run}/{total_runs}")
1191.          # Μηδενίζουμε την πρόοδο της Progress Bar για το νέο run.
1192.          self.progress_bar['value'] = 0
1193.          else:
1194.          # Σε άλλη περίπτωση εμφανίζουμε μήνιματος σφάλματος στην
κονσόλα και στο log widget.
1195.          log_entry = f"Error with message: Expected tuple of 2, got:
{type(msg_data)}, Value: {msg_data}"
1196.          self.log_message(log_entry)
1197.          print(log_entry) # στην κονσόλα
1198.
1199.          # Αν είναι μήνυμα τύπου "progress_update".
1200.          elif msg_type == "progress_update":
1201.          # Ενημέρωση progress bar μέσα στο RUN.
1202.          iteration = msg_data.get("iteration", 0)
1203.          max_iterations = msg_data.get("max_iterations", 1) # Αποφυγή
διαίρεσης με το 0.
1204.          alpha_fitness = msg_data.get("alpha_fitness", "N/A")
1205.          std_dev = msg_data.get("std_dev", "N/A")
1206.
1207.          # Ενημερώνουμε την ετικέτα με τον τρέχοντα αριθμό επανάληψης.
1208.          self.current_iter_label.config(text=f"{iteration}/{max_iterations}")
1209.
1210.          # Ενημερώνουμε την ετικέτα με το καλύτερο fitness.
```

```

1211.      # Αν το alpha_fitness είναι αριθμός (int ή float) τότε:
1212.      if isinstance(alpha_fitness, (int, float)):
1213.          # μορφοποιούμε την τιμή σε επιστημονική μορφή με 4 δεκαδικά
ψηφία.
1214.          fitness_text_value = f"{alpha_fitness:.4e}"
1215.      else:
1216.          # Αν δεν είναι αριθμός μετατρέπουμε σε string την υπάρχουσα
τιμή της μεταβλητής.
1217.          fitness_text_value = str(alpha_fitness)
1218.          # Ενημερώνουμε το label του fitness.
1219.          self.alpha_fitness_label.config(text=fitness_text_value)
1220.
1221.          # Ενημερώνουμε την ετικέτα με το την τυπική απόκλιση του fitness.
1222.          # Αν το std_dev είναι αριθμός (int ή float) τότε:
1223.          if isinstance(std_dev, (int, float)):
1224.              # Την μορφοποιούμε σε επιστημονική μορφή με 4 δεκαδικά
ψηφία.
1225.              deviation_text_value = f"{std_dev:.4e}"
1226.          else:
1227.              # Αν δεν είναι αριθμός το μετατρέπουμε σε string όπως είναι.
1228.              deviation_text_value = str(std_dev)
1229.
1230.          # Ενημερώνουμε το label της τυπικής απόκλισης.
1231.          self.std_dev_label.config(text=deviation_text_value)
1232.
1233.          # Υπολογίζουμε και ενημερώνουμε την μπάρα προόδου (0-100%).
1234.          if max_iterations > 0:
1235.              # Υπολογισμός προόδου με βάση και τη σύγκλιση
1236.              progress_iter = iteration / max_iterations
1237.
1238.              # Αν υπάρχει std_dev και είναι αριθμός, υπολογίζουμε και πρόοδο
σύγκλισης
1239.              if isinstance(std_dev, (int, float)) and std_dev > 0:
1240.                  try:
1241.                      min_std_dev_threshold = float(self.parameters_dict["min_std_dev"].get())
1242.                      convergence_progress = 1.0 - min(std_dev /
min_std_dev_threshold, 1.0)
1243.                  except:
1244.                      convergence_progress = 0.0
1245.              else:
1246.                  convergence_progress = 0.0
1247.
1248.              # Συνδυασμός επαναλήψεων και σύγκλισης για πιο ρεαλιστική
πρόοδο
1249.              combined_progress = (progress_iter + convergence_progress) / 2
1250.              self.progress_bar['value'] = combined_progress * 100
1251.
1252.

```

```

1253.      # Αν είναι ενεργοποιήσιμο το toggle button της καμπύλης σύγκλισης
           ενημέρωνουμε την καμπύλη
1254.      if self.enable_convergence_plot_var.get():
1255.      # Αν η καταλληλότητα δεν είναι None Και είναι σε κατάλληλο
           τύπο δεδομένων
1256.      if alpha_fitness is not None and isinstance(alpha_fitness,(int,
           float)):
1257.      # Ενημέρωνουμε τα values των κλειδιών του λεξικού που
           κρατάμε για τα δεδομένα της καμπύλης σύγκλισης
1258.      self.convergence_data['iterations'].append(iteration) # Αξονας X
           του Γραφήματος
1259.      self.convergence_data['fitness'].append(alpha_fitness) # Αξονας
           Y του Γραφήματος
1260.
1261.      # Διαβάζουμε την συχνότητα ενημέρωσης από το spinbox του
           GUI.
1262.      try:
1263.      # Λαμβάνουμε την τιμή του κλειδιού plot_update_freq του
           λεξικού με τις τιμές των παραμέτρων
1264.      # που έχει εισάγει ο Χρήστης στα πεδία εισόδου για την
           εκτέλεση του GWO
1265.      update_freq_str = self.parameters_dict["plot_update_freq"].get()
1266.      update_frequency = int(update_freq_str) # Μετατροπή σε int
1267.      # Σε περίπτωση πρόκλησης εξαίρεσης η default τιμή είναι 5.
1268.      except (ValueError, KeyError, AttributeError):
1269.      update_frequency = 5
1270.      # Συνθήκες Ενημέρωσης του Γραφήματος
1271.      update_conditions = (
1272.      iteration == 0 or # Στην 1η Επανάληψη
1273.      (iteration > 0 and iteration % update_frequency == 0) or # Κάθε
           N επαναλήψεις που έχει επιλέξει ο χρήστης στο Spinbox
1274.      (max_iterations > 0 and iteration == max_iterations-1 and itera-
           tion % update_frequency != 0) # Στην τελευταία Επανάληψη
1275.      )
1276.      # Αν οι συνθήκες ενημέρωσης είναι True:
1277.      if update_conditions:
1278.      #print(f"DEBUG: Updating Convergence Curve at iteration {it-
           eration} due to frequency {update_frequency}")
1279.      # Καλείται η μέθοδος update_convergence_plot() και
           σχεδιάζεται η Καμπύλη Σύγκλισης της Καταλληλότητας.
1280.      self.update_convergence_plot()
1281.
1282.      # Ενημέρωση Καμπυλών Χώρου Αναζήτησης.
1283.      elif msg_type == "spatial_plot_data":
1284.      # Αν είναι ενεργοποιήσιμο το toggle για την εμφάνιση των
           γραφημάτων.
1285.      if self.enable_spatial_live_update_var.get():
1286.      try:
1287.      # Ομοίως με προηγούμενως με την καμπύλη σύγκλισης

```

```
1288.         update_freq_str = self.parameters_dict["plot_update_freq"].get()
1289.         update_frequency = int(update_freq_str)
1290.     except (ValueError, KeyError, AttributeError):
1291.         update_frequency = 5
1292.         # Προσπαθεί να πάρει το value του key="iteration" απο το λεξικό
αν δεν τα καταφέρει θα θέσει την τιμή 0
1293.         iteration = msg_data.get("iteration",0)
1294.
1295.     try:
1296.         max_iters_str = self.parameters_dict["max_iter"].get()
1297.         max_iterations_for_plot_check = int(max_iters_str)
1298.         if max_iterations_for_plot_check <= 0:
1299.             max_iterations_for_plot_check = 1
1300.     except:
1301.         max_iterations_for_plot_check = 1
1302.
1303.         # Συνθήκες Ενημέρωσης του Γραφήματος
1304.         update_conditions = (
1305.             iteration == 0 or # Στην 1η Επανάληψη
1306.             (iteration > 0 and iteration % update_frequency == 0) or # Κάθε
Ν επαναλήψεις που έχει επιλέξει ο χρήστης στο Spinbox
1307.             (max_iterations > 0 and iteration == max_iterations-1 and itera-
tion % update_frequency != 0) # Στην τελευταία Επανάληψη
1308.         )
1309.         # Αν οι συνθήκες ενημέρωσης είναι True:
1310.         if update_conditions:
1311.             #print(f"DEBUG: Updating Spatial plot at iteration {iteration}
due to frequency {update_frequency}")
1312.
1313.             # Ελέγχουμε αν έχει δημιουργηθεί το widget notebook που θα
φιλοξενήσει τις καρτέλες των γραφικών παραστάσεων
1314.             # αν δεν υπάρχει τότε συνεχίζουμε στην επόμενη επανάληψη
του for για την κατανάλωση του επόμενου μηνύματος της ουράς.
1315.             if not hasattr(self, "plot_notebook") or self.plot_notebook is
None:
1316.                 continue
1317.             try:
1318.                 # Αν υπάρχει ακόμα το Notebook και υπάρχει ενεργή
καρτέλα τότε:
1319.                 if self.plot_notebook.wininfo_exists() and self.plot_note-
book.select():
1320.                     # Λαμβάνουμε το Index του στοιχείου της λίστας που
αντιστοιχεί στο λεξικό με τις ρυθμίσεις της επιλεγμένης ενεργής καρτέλας
1321.                     current_tab_index = self.plot_notebook.in-
dex(self.plot_notebook.select())
1322.                     # Λαμβάνουμε το κλειδί απο μία λίστα που περιέχει λεξικά
1323.                     # με ρυθμίσεις για τα τρία γραφήματα και άρα για τις 3
καρτέλες
```

```

1324.         active_tab_key = self.plot_configs_list[current_tab_in-
dex][["key"]]
1325.         else:
1326.             active_tab_key = None
1327.         except(tk.TclError, IndexError, AttributeError):
1328.             active_tab_key = None
1329.
1330.         if active_tab_key == "3d_view":
1331.             #print(f"DEBUG: Updating 3D Plot at iteration: {iteration}.")
1332.             self.update_3d_plot_content(msg_data)
1333.
1334.         elif active_tab_key == "top_view":
1335.             #print(f"DEBUG: Updating Top View Plot at iteration: {iter-
ation}.")
1336.             self.update_top_view_plot_content(msg_data)
1337.
1338.         elif msg_type == "run_completed_stats":
1339.             run_exec_time = msg_data.get("gwo_time", 0.0)
1340.             if isinstance(run_exec_time, (int, float)):
1341.                 # Σωρεύουμε τον χρόνο εκτέλεσης του κάθε RUN σε μία
μεταβλητή σωρευτή
1342.                 self.cumulative_execution_time += run_exec_time
1343.
1344.                 time_text_value = f"{self.cumulative_execution_time:.4f}"
1345.                 # Ενημερώνουμε το Label του Χρόνου για να ενημερωθεί ο
Χρήστης.
1346.                 self.exec_time_label.config(text=time_text_value)
1347.
1348.                 # Αν το μήνυμα είναι τύπου "all_runs_finished", το οποίο στέλνεται
οταν ολοκληρωθεί η εκτέλεση του αλγορίθμου
1349.                 # ή ο χρήστης έχει πατήσει το κουμπί STOP τότε:
1350.                 elif msg_type == "all_runs_finished":
1351.                     # Όλα τα runs ολοκληρώθηκαν ή σταμάτησαν
1352.                     self.log_message("--- All GWO RUNS completed or stopped. ---")
1353.                     # Ενεργοποιούμε και απενεργοποιούμε τα κουμπιά αναλόγως
1354.                     self.run_button.config(state=NORMAL)
1355.                     self.stop_button.config(state=DISABLED)
1356.                     self.progress_bar['value'] = 100
1357.                     # Αποθηκεύουμε τα δεδομένα του τελευταίου RUN του αλγορίθμου
1358.                     final_results_data = msg_data
1359.
1360.                     # Τελική ενημέρωση του γραφήματος σύγκλισης
1361.                     if self.enable_convergence_plot_var.get():
1362.                         #print("DEBUG: Final update for convergence plot.")
1363.                         self.update_convergence_plot()
1364.                     # Τελική ενημέρωση των γραφημάτων 3D ή Top View αναλόγως της
ενεργής καρτέλας.
1365.                     #if self.enable_spatial_live_update_var.get() and final_results_data:
1366.                         #print("DEBUG: Attempting final update for active spatial plot.")

```

```

1367.         #self.update_3d_plot_content(msg_data)
1368.         #self.update_top_view_plot_content(msg_data)
1369.     except Exception as e:
1370.         # Εμφανίζουμε τυχόν λάθη που προέκυψαν κατά την επεξεργασία
1371.         self.log_message(f'Error processing GUI queue: {e}')
1372.     finally:
1373.         # Η επόμενη κλήση της μεθόδου θα γίνει σε 100msec προκειμένου να
ελέγχουμε
1374.         # συνέχεια για τυχόν νέα μηνύματα απο το νήμα του GWO.
1375.         self.after(100, self.process_gui_queue)
1376.
1377.
1378.     def gui_callback_from_thread(self, msg_type, data):
1379.         """
1380.         Η συνάρτηση στέλνει μηνύματα απο το νήμα εκτέλεσης του GWO στο νήμα
του GUI
1381.         με χρήση μιας thread safe ουρας για να αποφύγουμε race conditions μεταξύ
των δύο νημάτων.
1382.
1383.         Args:
1384.             msg_type (str): τύπος μηνύματος (π.χ. "log").
1385.             data (any): Δεδομένα μηνύματος.
1386.         """
1387.         self.gui_queue.put((msg_type, data))
1388.
1389.     def stop_gwo(self):
1390.         """
1391.         Η συνάρτηση στέλνει σήμα διακοπής στον αλγόριθμο GWO και
απενεργοποιεί το κουμπί Stop.
1392.         """
1393.         # Αν το thread του GWO τρέχει:
1394.         if self.gwo_thread and self.gwo_thread.is_alive():
1395.             self.log_message("--- Stop signal sent. Waiting for current GWO operations
to complete... ---")
1396.             # το stop_event γίνεται True, η εκτέλεση του αλγορίθμου θα διακοπεί.
1397.             self.stop_event.set()
1398.             # Απενεργοποιούμε το κουμπί STOP
1399.             self.stop_button.config(state=DISABLED)
1400.
1401.
1402.     #7. Utility Μέθοδοι GUI
=====
=====
1403.
1404.     def log_message(self, message):
1405.         """
1406.         Εμφανίζει ένα μήνυμα στο text widget στο Log_frame.
1407.
1408.         Args:

```

```

1409.         message(str): Το μήνυμα που θα εμφανιστεί στο log.
1410.
1411.         """
1412.         # Ξεκλειδώνουμε προσωρινά το Text widget.
1413.         self.log_text.config(state=NORMAL)
1414.         # Προσθέτουμε το μήνυμα.
1415.         self.log_text.insert(END, str(message) + "\n")
1416.         # Αυτόματη κύλιση προς τα κάτω για να φαίνεται το τελευταίο μήνυμα.
1417.         self.log_text.see(END)
1418.         # Κλειδώνουμε ξανά για να μην μπορεί να το επεξεργαστεί ο χρήστης.
1419.         self.log_text.config(state=DISABLED)
1420.         # Γρήγορη ανανέωση για να φανεί το μήνυμα.
1421.         self.update_idletasks()
1422.
1423.     # 8. Μέθοδος Χειρισμού Κλεισίματος Παραθύρου
=====
1424.
1425.     def on_closing(self):
1426.         """
1427.         Η συνάρτηση χειρίζεται το κλείσιμο του παραθύρου της εφαρμογής.
1428.         Αν ο GWO τρέχει τότε προσπαθεί να προσπαθήσει πρώτα να τον σταματή-
1429.         σει,
1430.         εν συνεχεία κλείνει όλα τα γραφήματα Matplotlib και κλείνει το κυρίως πα-
1431.         ράθυρο της εφαρμογής.
1432.         """
1433.         # Ελέγχουμε αν το νήμα του GWO υπάρχει και είναι ενεργό.
1434.         if self.gwo_thread and self.gwo_thread.is_alive():
1435.             self.log_message("Window closing: Attempting to stop GWO thread...")
1436.             # Θέτουμε το stop_event σε True για να διακοπεί η εκτέλεση του GWO.
1437.             self.stop_event.set()
1438.             # Περιμένουμε για 1 sec να σταματήσει το thread και να επιστρέψει τυχόν
1439.             αποτελέσματα.
1440.             self.gwo_thread.join(timeout=1.0)
1441.             # Αν το νήμα τρέχει ακόμα ενημερώνουμε τον χρήστη με ένα μήνυμα
1442.             if self.gwo_thread.is_alive():
1443.                 self.log_message("Warning: GWO thread still running...")
1444.
1445.         # Κλείνουμε τα παράθυρα όλων των γραφημάτων.
1446.         plt.close("all")
1447.         # Κλείνουμε το παράθυρο της εφαρμογής.
1448.         self.destroy()
1449.
1450.     if __name__ == "__main__":
1451.         # Διασφαλίζουμε ότι το Matplotlib θα συνεργαστεί σωστά με το Tkinter
1452.         plt.switch_backend('TkAgg')
1453.         app = GWOApp()
1454.         app.mainloop()

```

A-2 Αρχείο main.py

```
1. #-*- coding: utf-8 -*-
2. """
3. @Student: Christos Smarlamakis
4. std147661@ac.eap.gr, christossmarlamakis@gmail.com / 6945830515
5. """
6.
7. import csv
8. import os
9. from benchmark_function import BenchmarkFunction
10. from gwo import GWO
11.
12.
13. # Ορισμός παραμέτρων
14. benchmark_functions_names_list = ["ackley", "griewank", "rastrigin", "rosenbrock",
15. "sphere", "dejongF5_foxholes", "goldstein-price"]
16. #benchmark_functions_names_list = ["dejongF5_foxholes", "goldstein-price"]
17. dimensions_number = 30
18. wolves_number = 60
19. max_iteration_num = 500
20. min_fitness_standard_deviation = 1e-10
21. runs = 30
22. plot_graph_enabled = 0 # True/1 ή False/0 για εμφάνιση ή μή του γραφήματος.
23. # Δημιουργία φακέλου αποθήκευσης αν δεν υπάρχει
24. results_folder = f"cli_results\\results_with_{wolves_number}_agents_cli"
25. os.makedirs(results_folder, exist_ok=True)
26.
27. # Εκτελούμε τον αλγόριθμο για όλες τις συναρτήσεις δοκιμής στην σειρά και
28. αποθηκεύουμε τα αποτελέσματα για στατιστική ανάλυση σε αρχεία csv.
29. for benchmark_function_name in benchmark_functions_names_list:
30.     # Αν η συνάρτηση είναι από τις dejongF5_foxholes ή goldstein-price, ορίζουμε τις
31.     # διαστάσεις σε 2.
32.     if benchmark_function_name in ["dejongF5_foxholes", "goldstein-price"]:
33.         dimensions = 2
34.     else:
35.         dimensions = dimensions_number
36.     benchmark_function = BenchmarkFunction(benchmark_function_name,
37. dimensions) #Δημιουργούμε ένα αντικείμενο BenchmarkFunction
38.     results_list = []
39.     print(f"{benchmark_function_name} function:")
40.     for i in range(runs):
41.         print(f"Run {i+1}/{runs}")
```

```
42.      #Εκτέλεση του Αλγορίθμου GWO με τις επιλεχθείσες παραμέτρους και
επιστροφή των αποτελεσμάτων στο λεξικό results_dictionary
43.      results_dictionary = GWO(wolves_number, max_iteration_num,
min_fitness_standard_deviation, benchmark_function, plot_graph_enabled)
44.      results_dictionary['run'] = i+1
45.      results_list.append(results_dictionary)
46.
47.      # Ορισμός διαδρομής αποθήκευσης αρχείου
48.      csv_filename = os.path.join(results_folder,
f"gwo_results_{benchmark_function_name}.csv")
49.
50.      # Αποθήκευση των αποτελεσμάτων σε CSV.
51.      with open(csv_filename, mode="w", newline="") as file:
52.          writer = csv.DictWriter(file, fieldnames = ['run', 'min_fitness', 'mean_fitness',
'alpha_wolf_position', 'alpha_wolf_fitness', 'global_minimum_value',
'fitness_standard_deviation', 'positional_deviation', 'iterations', 'gwo_time'], delimiter=';')
53.          writer.writeheader()
54.          writer.writerows(results_list)
55.
56.      print(f"Results saved in {csv_filename}\n")
```

A-3 Αρχείο gwo.py

```
1. #-*- coding: utf-8 -*-
2. """
3. @Student: Christos Smarlamakis
4. std147661@ac.eap.gr, christossmarlamakis@gmail.com / 6945830515
5. """
6.
7. import numpy as np
8. from scipy.stats import uniform
9. import concurrent.futures as cf
10. from time import perf_counter
11. import plot_graph as plot
12.
13.
14. def find_best_three_wolves(wolves_positions_array, benchmark_function, iteration_counter, plot_graph_enabled, gui_callback=None, stop_event=None):
15.     """
16.     Η συνάρτηση εντοπίζει απο το σύνολο των πρακτόρων τους τρεις καλύτερους με βάση την τιμή της καταλληλότητας.
17.
18.     Αρχικά δέχεται ως ορίσματα έναν πίνακα με τις θέσεις όλων των πρακτόρων, την συνάρτηση αξιολόγησης, τον αριθμό της τρέχουσας επανάληψης
19.     και μία σημαία για την εμφάνιση ή μη γραφικής παράστασης. Επιστρέφει ένα λεξικό με τις θέσεις και τις καταλληλότητες των ηγετών λύκων,
20.     την τυπική απόκλιση των καταλληλοτήτων και τέλος τις τιμές των fitness όλων των λύκων.
21.
22.     Args:
23.         wolves_positions_array (numpy.ndarray): Πίνακας NumPy διαστάσεων (N, D) με τις θέσεις των N λύκων στον χώρο αναζήτησης D διαστάσεων.
24.         benchmark_function (object): Η συνάρτηση αξιολόγησης.
25.         iteration_counter (int): Ο αριθμός της τρέχουσας επανάληψης.
26.         plot_graph_enabled (bool): Σημαία για την εμφάνιση γραφήματος ανα 20 επαναλήψεις αν το πρόβλημα είναι 2D.
27.
28.     Returns:
29.         dict: Λεξικό με τους τρεις καλύτερους λύκους (θέση και fitness).
30.         float: Τυπική απόκλιση των τιμών καταλληλοτήτων του πληθυσμού που αποτελεί δείκτη σύγκλιση του αλγορίθμου στο ολικό βέλτιστο.
31.         numpy.ndarray: Πίνακας NumPy με τις τιμές καταλληλότητας όλων των λύκων.
32.     """
33.     # Αρχικοποίηση του λεξικού για τους καλύτερους λύκους alpha, beta και delta.
34.     # Τα κλειδιά για τους λύκους είναι το 0 : alpha, 1 : beta, 2 : delta.
35.     # Η τιμή του κάθε κλειδιού είναι μία λίστα με δύο στοιχεία, το διάνυσμα της θέσης του λύκου και η καταλληλότητα του (αρχική τιμή άπειρο-inf).
36.     best_wolves_dictionary = {
```

```

37.      0 : [np.zeros(wolves_positions_array.shape[1]), float("inf")],
#[[0.0,...,0.0],inf]
38.      1 : [np.zeros(wolves_positions_array.shape[1]), float("inf")],
#[[0.0,...,0.0],inf]
39.      2 : [np.zeros(wolves_positions_array.shape[1]), float("inf")]
#[[0.0,...,0.0],inf]
40.      }
41.
42.
43.  # Υπολογισμός καταλληλοτήτων των λύκων με παράλληλη εκτέλεση χρησιμοποι-
ούμε το with..as.. για αποφυγή διαρροών μνήμης και αυτόματο κλείσιμο του executor.
44.  with cf.ThreadPoolExecutor() as executor:
45.      # Με την Map() εκτελείται η συνάρτηση calculate_fitness() σε κάθε στοιχείο του
wolves_positions_array.
46.      # ακολούθως με την list() μετατρέπεται σε λίστα ο iterator που επιστρέφεται απο
την Map() με τα δεδομένα.
47.      wolves_fitness_list = list(executor.map(benchmark_function.calculate_fitness,
wolves_positions_array))
48.
49.  # Μετατροπή της λίστας με τις καταλληλότητες των λύκων σε numpy array για ευ-
κολία στους υπολογισμούς.
50.  wolves_fitness_numpy_array = np.array(wolves_fitness_list)
51.  #Η Np.argsort() επιστρέφει μία λίστα με τους δείκτες της λίστας που αν αντιστοι-
χηθούν σε τιμές θα προκύψει η αυξουσα ταξινομημένη διάταξη των τιμών.
52.  sorted_indices = np.argsort(wolves_fitness_numpy_array)
53.  #Πραγματοποιείται λήψη με indexing απο την λίστα sorted_indices[] των τριών
πρώτων στοιχείων που αποτελούν και τα Index του πίνακα wolves_positions_array[] με
τις μικρότερες τιμές του fitness.
54.  #και άρα τις καταλληλότητες των τριών ηγετών λύκων. Ακολούθως αποθηκεύουμε
σε ένα λεξικό τις θέσεις και τις καταλληλότητες των τριών καλύτερων λύκων.
55.  for i in range(3):
56.      best_wolves_dictionary[i] = [wolves_positions_array[sorted_indices[i]],
wolves_fitness_numpy_array[sorted_indices[i]]]
57.
58.  #Σε περίπτωση που δεν γίνεται εκτέλεση του αλγορίθμου απο το Γραφικό Περιβάλ-
λον σχεδιάζουμε την γραφική παράσταση με τις θέσεις των λύκων.
59.  if gui_callback is None:
60.      # Αν το πρόβλημα είναι δισδιάστατο (2D) εμφανίζεται ανα 20 επαναληψεις το
γράφημα της συνάρτησης με τις θέσεις των λύκων και την θέση του ολικού βέλτιστου (ε-
λαχίστου).
61.      if benchmark_function.dimensions == 2 and iteration_counter % 2 == 0 and
plot_graph_enabled == True:
62.          plot.plot_graph(benchmark_function, wolves_positions_array, wolves_fit-
ness_numpy_array, iteration_counter)
63.          #Αφαιρώντας το σχόλιο απο την κάτωθι γραμμή δύναται να απεικονιστεί και η
κάτοψη του γραφήματος.
64.          #plot.plot_top_view_graph(benchmark_function,wolves_positions_array,itera-
tion_counter).
65.

```

```
66. # Υπολογισμός της τυπικής απόκλισης της καταλληλότητας των λύκων.
67. fitness_standard_deviation = np.std(wolves_fitness_numpy_array)
68. #Επιστροφή λεξικού ηγετών λύκων, τυπικής απόκλισης καταλληλοτήτων και πί-
νακα καταλληλοτήτων όλων των λύκων.
69. return best_wolves_dictionary, fitness_standard_deviation, wolves_fit-
ness_numpy_array
70.
71. def calculate_new_position(best_wolves_dictionary, wolf_position, a, dimension):
72.     """
73.     Η μέθοδος υπολογίζει τις νέες θέσεις των λύκων με βάσει τις θέσεις των τριών κα-
λύτερων λύκων.
74.
75.     Υπολογίζει τους τυχαίους συντελεστές r1,r2 και τις παραμέτρους A, C για να καθο-
ρίσει την σύγκλιση ή όχι των υπόλοιπων λύκων.
76.     προς τους τρεις ηγέτες λύκους, εν συνεχεία υπολογίζει τη νέα θέση του κάθε λύκου
στη τρέχουσα διάσταση ως το μέσο όρο των θέσεων των τριών ηγετών λύκων.
77.
78.     Args:
79.         best_wolves_dictionary (dict): Λεξικό με θέσεις και fitness των 3 καλύτερων λύ-
κων.
80.         wolf_position (float): Η τρέχουσα θέση του λύκου στη συγκεκριμένη διάσταση.
81.         a (float): Συντελεστής a που ισορροπεί την εξερεύνηση και την εκμετάλλευση
του GWO.
82.         dimension (int): Η τρέχουσα διάσταση.
83.
84.     Returns:
85.         float: Η νέα θέση του λύκου στη συγκεκριμένη διάσταση.
86.     """
87.     # Για κάθε διάσταση υπολογίζονται οι δύο τυχαία συντελεστές r1,r2 με ομοιό-
μορφη κατανομή που καθορίζουν την ένταση και την κατεύθυνση της αναζήτησης - εξε-
ρεύνησης (exploration).
88.     r1, r2 = uniform.rvs(size=2) #Επιστροφή δύο τυχαίων αριθμών στο [0,1] με ομοιό-
μορφη κατανομή (ισοπιθανοί).
89.     A = 2 * a * r1 - a #Υπολογισμός παραμέτρου A που σχετίζεται με την απόφαση συ-
νέχισης αναζήτησης ή σύγκλισης του GWO.
90.     # Υπολογισμός της παραμέτρου C που σχετίζεται με την σύγκλιση σε υποψήφια
λύση.
91.     C = 2 * r2
92.
93.     #Υπολογισμός της απόστασης D μεταξύ του τρέχοντα λύκου και των τριών καλύ-
τερων λύκων.
94.     # D = |C * x_best_wolf[v][0][problems_variables] - x_current_wolf|
95.     D_alpha = abs(C * best_wolves_dictionary[0][0][dimension] - wolf_position)
96.     D_beta = abs(C * best_wolves_dictionary[1][0][dimension] - wolf_position)
97.     D_delta = abs(C * best_wolves_dictionary[2][0][dimension] - wolf_position)
98.
99.     # Ενημέρωση του αντίστοιχου στοιχείου new_position_temp_vector με την υπολο-
γισμένη τιμή.
```

```

100. # Υπολογισμός της νέας συντεταγμένης της τρέχουσας διάστασης βάσει του τύπου
X[dimension] = best_wolf_position - A * D.
101. position_on_current_dimension_based_on_alpha_wolf = best_wolves_dictionary[0][0][dimension] - A * D_alpha
102. position_on_current_dimension_based_on_beta_wolf = best_wolves_dictionary[1][0][dimension] - A * D_beta
103. position_on_current_dimension_based_on_delta_wolf = best_wolves_dictionary[2][0][dimension] - A * D_delta
104. new_position_on_current_dimension = (position_on_current_dimension_based_on_alpha_wolf + position_on_current_dimension_based_on_beta_wolf + position_on_current_dimension_based_on_delta_wolf) / 3
105. #Επιστροφή της νέας θέση του τρέχοντος λύκου για την τρέχουσα διάσταση.
106. return new_position_on_current_dimension
107.
108. def GWO(wolves_number, max_iteration_num, min_fitness_standard_deviation, benchmark_function, plot_graph_enabled, gui_callback=None, stop_event=None):
109.     """
110.     Η συνάρτηση υλοποιεί τον αλγόριθμο GWO για την εύρεση του ολικού βέλτιστου ενός δεδομένου προβλήματος.
111.
112.     Αρχικά εκτελείται η αρχικοποίηση των θέσεων των πρακτόρων αναζήτησης (λύκων), εν συνεχεία σε επαναληπτικά βήματα προσαρμόζει συνεχώς
113.     τις θέσεις των λύκων με βάση τις θέσεις των τριών ηγετών λύκων, μειώνει την παράμετρο a που ισοροπεί τις φάσεις της εξερεύνησης και της εκμετάλευσης
114.     της ευρεθείσας υποψήφιας λύσης. Τέλος Επιστρέφονται τα αποτελέσματα με την θέση του ανευρεθέντος βέλτιστου κ.α.
115.
116.     Args:
117.         wolves_number (int): Το μέγεθος του πληθυσμού των λύκων.
118.         max_iteration_num (int): Ο μέγιστος αριθμός επαναλήψεων για τον αλγόριθμο (Κριτήριο Τερματισμού A).
119.         min_fitness_standard_deviation (float): Η ελάχιστη τυπική απόκλιση των fitness για να σταματήσει η αναζήτηση (Κριτήριο Τερματισμού B).
120.         benchmark_function (BenchmarkFunction): Η συνάρτηση benchmark για την οποία εκτελείται η βελτιστοποίηση.
121.         plot_graph_enabled (bool): Σημαία για την απεικόνιση της γραφικής παράστασης της διαδικασίας αναζήτησης.
122.
123.     Returns:
124.         dict: Λεξικό με τα αποτελέσματα της εκτέλεσης του αλγορίθμου.
125.         Το λεξικό περιέχει τα εξής πεδία:
126.         - "global_minimum_value": Η τιμή του ολικού ελάχιστου της συνάρτησης benchmark.
127.         - "mean_fitness": Ο μέσος όρος της fitness των λύκων.
128.         - "min_fitness": Η ελάχιστη fitness από όλους τους λύκους.
129.         - "fitness_standard_deviation": Η τυπική απόκλιση των fitness.
130.         - "positional_deviation": Η ευκλείδεια απόσταση της θέσης του άλφα λύκου από το ολικό ελάχιστο.
131.         - "iterations": Ο αριθμός των επαναλήψεων του αλγορίθμου.

```

```
132.         - "gwo_time": Ο χρόνος εκτέλεσης του αλγορίθμου σε δευτερόλεπτα.
133.     """
134.     #Καταγραφή του χρόνου έναρξης της εκτέλεσης του αλγορίθμου.
135.     start = perf_counter()
136.
137.     if wolves_number < 3:
138.         if gui_callback:
139.             return {"error": "At least 3 Wolves required (alpha, beta, delta.", "gwo_time"
140. : 0}
141.         else:
142.             raise ValueError("At least 3 Wolves required (alpha, beta, delta.")
143.     lower_bound, upper_bound = benchmark_function.get_fbounds()
144.     variables_number = benchmark_function.dimensions
145.
146.     # Δημιουργία ενός NumPy πίνακα διαστάσης wolves_number * variables_number
147.     # ο οποίος περιέχει τις θέσεις των λύκων.
148.     wolves_pos_array = np.random.uniform(low=lower_bound, high=upper_bound,
149. size=(wolves_number, variables_number))
150.     # Αρχικοποίηση της παραμέτρου a που σχετίζεται με την ισορροπία μεταξύ των
151.     # φάσεων εξερεύνησης και εκμετάλευσης.
152.     a_factor = 2
153.     # Μετρητής επαναλήψεων του αλγορίθμου.
154.     iteration_counter = 0
155.
156.     # Κλήση της find_best_wolves() για την εύρεση των λύκων α,β,δ και επιστροφή
157.     # τους μαζί με την απόκλιση του fitness όλων των λύκων.
158.     best_wolves_dictionary, fitness_standard_deviation, wolves_fitness_numpy_array
159.     = find_best_three_wolves(wolves_pos_array, benchmark_function, iteration_counter,
160. plot_graph_enabled, gui_callback, stop_event)
161.
162.     # Εκτέλεση της Αναζήτησης για έναν συγκεκριμένο αριθμό επαναλήψεων
163.     # (max_iteration_num) ή έως ότου η τυπική απόκλιση των καταλληλοτήτων των λύκων
164.     # μειωθεί.
165.     # περισσότερο απο κάποιο κατώφλι (fitness_standart_deviation)
166.     while iteration_counter < max_iteration_num and fitness_standard_deviation >
167. min_fitness_standard_deviation:
168.
169.         if stop_event and stop_event.is_set():
170.             if gui_callback:
171.                 gui_callback("log", f"GWO: forcing the algorithm to stop at iteration {iteration_counter} due to stop signal.")
172.             break
173.
174.         # Σε κάθε επανάληψη του αλγορίθμου εκτελείται μία ανακύκλωση στον πληθυσμό των λύκων.
175.         for wolf in range(wolves_number):
176.             if stop_event and stop_event.is_set(): break
177.
178.
```

```

169.         # Για κάθε λύκο και για κάθε μεταβλητή (διάσταση).
170.         for every_dimension in range(variables_number):
171.             # καλείται η calculate_new_position() και υπολογίζεται η νέα θέση του λύ-
172.             # κου με βάση τις θέσεις των λύκων ηγετών ως ο μέσος όρος αυτών.
173.             new_position = calculate_new_position(best_wolves_dictionary,
174.             wolves_pos_array[wolf, every_dimension], a_factor, every_dimension)
175.             # Επίσης για κάθε συντεταγμένη της νέας θέσης καλείται η np.clip() η ο-
176.             # ποία ελέγχει και αν χρειάζεται μεταφέρει εντός ορίων συνάρτησης την νέα θέση.
177.             wolves_pos_array[wolf, every_dimension] = np.clip(new_posi-
178.             tion, lower_bound, upper_bound)
179.
180.             # Η παράμετρος α μειώνεται γραμμικά σύμφωνα με τον τύπο της θεωρίας σε
181.             # κάθε επανάληψη (σταδιακή μετάβαση από την εξερεύνηση στην εκμετάλλευση).
182.             a_factor = 2 - iteration_counter * (2 / max_iteration_num)
183.             # Υπολογισμός εκ νέου των τριών ηγετών λύκων.
184.             best_wolves_dictionary, fitness_standard_deviation, wolves_fitness_numpy_ar-
185.             ray = find_best_three_wolves(wolves_pos_array, benchmark_function, iteration_counter,
186.             plot_graph_enabled, gui_callback, stop_event)
187.
188.             if gui_callback:
189.                 alpha_fitness_val = float('inf')
190.
191.                 if 0 in best_wolves_dictionary and best_wolves_dictionary[0] and
192.                 len(best_wolves_dictionary[0]) > 1:
193.                     alpha_fitness_val = best_wolves_dictionary[0][1]
194.
195.             # Στέλνουμε στο GUI ένα λεξικό με τα δεδομένα για ενημέρωση των στοι-
196.             # χείων του GUI
197.             gui_callback("progress_update", {
198.                 "iteration": iteration_counter,
199.                 "max_iterations": max_iteration_num,
200.                 "alpha_fitness": best_wolves_dictionary[0][1],
201.                 "std_dev": fitness_standard_deviation,
202.                 #"convergence_data": (iteration_counter, best_wolves_dictionary[0][1])
203.             })
204.             # Σε περίπτωση προβλήματος με 2 μεταβλητές στέλνουμε και τα απαραίτητα
205.             # δεδομένα πάλι σε λεξικό
206.             # για την σχεδίαση των γραφημάτων
207.             if benchmark_function.dimensions == 2 and plot_graph_enabled:
208.                 # print(f"DEBUG GWO: Sending spatial_plot_data at iteration: {itera-
209.                 # tion_counter}")
210.                 gui_callback("spatial_plot_data", {
211.                     "iteration": iteration_counter,
212.                     "wolves_positions": wolves_pos_array.copy(),
213.                     "wolves_fitness": wolves_fitness_numpy_array.copy(),
214.                     #"benchmark_function": benchmark_function,
215.                 })
216.             # Αύξηση του μετρητή επαναλήψεων .
217.             iteration_counter += 1

```

```
207.
208.     end = perf_counter()
209.     gwo_time = end - start
210.
211.     #Συντεταγμένες Ακριβούς Ολικού Ελαχίστου.
212.     global_minimum_point_coords = np.array(benchmark_function.get_fmini-
mum_point())
213.     #Συντεταγμένες σημείου της τρέχουσας ευρεθείσας λύσης (Συντεταγμένες Θέσης
Λύκου Alpha)
214.     alpha_wolf_position = np.array(best_wolves_dictionary[0][0])
215.     # Υπολογισμός της απόκλισης από το ολικό ελάχιστο με υπολογισμό της ευκλεί-
διας απόστασης του άλφα λύκου απο το ολικό ελάχιστο.
216.     positional_deviation = np.linalg.norm(alpha_wolf_position - global_mini-
mum_point_coords)
217.
218.     if 0 in best_wolves_dictionary and best_wolves_dictionary[0]:
219.         if best_wolves_dictionary[0][0] is not None:
220.             alpha_wolf_position = np.array(best_wolves_dictionary[0][0])
221.         if best_wolves_dictionary[0][1] is not None:
222.             alpha_wolf_fitness_final = best_wolves_dictionary[0][1]
223.
224.     positional_deviation = np.linalg.norm(alpha_wolf_position - global_mini-
mum_point_coords) if not np.any(np.isnan(alpha_wolf_position)) else np.nan
225.
226.     # Έλεγχος αν το array δεν είναι άδειο
227.     if len(wolves_fitness_numpy_array) > 0:
228.         # Επιλογή μόνο των πεπερασμένων τιμών (όχι NaN ή Inf)
229.         finite_values = wolves_fitness_numpy_array[np.isfinite(wolves_fit-
ness_numpy_array)]
230.
231.         # Έλεγχος αν υπάρχουν πεπερασμένες τιμές
232.         if len(finite_values) > 0:
233.             min_fitness_overall = np.min(finite_values)
234.             mean_fitness_overall = np.mean(finite_values)
235.         else:
236.             min_fitness_overall = np.inf
237.             mean_fitness_overall = np.inf
238.     else:
239.         min_fitness_overall = np.inf
240.         mean_fitness_overall = np.inf
241.
242.     if fitness_standard_deviation is None:
243.         fitness_std = np.inf
244.     else:
245.         fitness_std = fitness_standard_deviation
246.
247.     # Δημιουργία λεξικού με τα αποτελέσματα.
248.     results = {
249.         "global_minimum_value": benchmark_function.get_fminimum_value(),
```

```
250.     "mean_fitness": mean_fitness_overall,
251.     "min_fitness": min_fitness_overall, # Η συνολική καλύτερη fitness που βρέθηκε
252.     "alpha_wolf_fitness": alpha_wolf_fitness_final, # Η fitness του alpha στο τέλος
253.     "alpha_wolf_position": alpha_wolf_position.tolist(), # Μετατροπή σε λίστα
254.     "fitness_standard_deviation" : fitness_std,
255.     "positional_deviation": positional_deviation,
256.     "iterations": iteration_counter, # Ο αριθμός των επαναλήψεων που
ολοκληρώθηκαν
257.     "gwo_time": gwo_time
258. }
259.
260. #Επιστροφή του λεξικού των αποτελεσμάτων.
261. return results
262.
```

A-4 Αρχείο benchmark_function.py

```
1. #-*- coding: utf-8 -*-
2. """
3. @Student: Christos Smarlamakis
4. std147661@ac.eap.gr, christossmarlamakis@gmail.com / 6945830515
5. """
6.
7. import numpy as np
8.
9. class BenchmarkFunction:
10.
11.     """
12.     Η κλάση αυτή δημιουργεί αντικείμενα συναρτήσεων benchmark (δοκιμής).
13.     """
14.
15.     def __init__(self, name, dimensions=2):
16.         """
17.         Αρχικοποιητής με ορισμα το όνομα της συνάρτησης δοκιμής
18.         """
19.         self.name = name #Πεδίο για το όνομα της συνάρτησης
20.         self.dimensions = dimensions #Πεδίο για το πλήθος των μεταβλητών του
21.         self.lower_bound, self.upper_bound = self.get_fbounds() #πεδία για τα όρια της
22.         #προβλήματος / διαστάσεις προβλήματος
23.
24.     def get_fbounds(self):
25.         """
26.         Η get_bounds() επιστρέφει τα όρια της τρέχουσας συνάρτησης δοκιμής.
27.         """
28.         if self.name == "ackley": #F10
29.             return -32, +32
30.         elif self.name == "rastrigin": #F9
31.             return -5.12, +5.12
32.         elif self.name == "rosenbrock": #F5
33.             return -30, +30
34.         elif self.name == "sphere": #F1
35.             return -100, +100
36.         elif self.name == "griewank": #F11
37.             return -600, +600
38.         elif self.name == "dejongF5_foxholes": #F14
39.             return -65.536, +65.536
40.         elif self.name == "goldstein-price": #F18
41.             return -2, +2
42.         else:
43.             raise ValueError("Unknown benchmark function")
44.
```

```
45. def get_fminimum_value(self):
46.     """
47.     Η get_actual_minimum() επιστρέφει το πραγματικό ελάχιστο της τρέχουσας
    συνάρτησης δοκιμής
48.     """
49.     if self.name == "ackley":
50.         return 0
51.     elif self.name == "griewank":
52.         return 0
53.     elif self.name == "rastrigin":
54.         return 0
55.     elif self.name == "rosenbrock":
56.         return 0
57.     elif self.name == "sphere":
58.         return 0
59.     elif self.name == "dejongF5_foxholes":
60.         return 1
61.     elif self.name == "goldstein-price":
62.         return 3
63.     else:
64.         raise ValueError("Unknown benchmark function")
65.
66. def get_fminimum_point(self):
67.     """
68.     Η get_actual_minimum() επιστρέφει το πραγματικό ελάχιστο της τρέχουσας
    συνάρτησης δοκιμής
69.     """
70.     if self.name == "ackley":
71.         return tuple(0 for _ in range(self.dimensions)) #(0,...0)
72.     elif self.name == "griewank":
73.         return tuple(0 for _ in range(self.dimensions)) #(0,...0)
74.     elif self.name == "rastrigin":
75.         return tuple(0 for _ in range(self.dimensions)) #(0,...0)
76.     elif self.name == "rosenbrock":
77.         return tuple(1 for _ in range(self.dimensions)) #(1,...0)
78.     elif self.name == "sphere":
79.         return tuple(0 for _ in range(self.dimensions)) #(0,...0)
80.     elif self.name == "dejongF5_foxholes":
81.         return tuple(-32 for _ in range(2)) #(-32,-32)
82.     elif self.name == "goldstein-price":
83.         return (0, -1) #(0,-1)
84.     else:
85.         raise ValueError("Unknown benchmark function")
86.
87. def calculate_fitness(self, x):
88.
89.     """
90.     Η συνάρτηση calculate_fitness() παίρνει ως όρισμα μία λίστα/πίνακα με τις
    συντεταγμένες της θέσης
```

```

91.     και υπολογίζει την καταλληλότητα του λύκου (πράκτορα αναζήτησης)
92.     """
93.     n = len(x)
94.     if self.name == "ackley": #F10 - Multimodal Function
95.         #  $f(x) = -20 \cdot \exp(-0.2 \cdot \sqrt{1/n \cdot \sum_i (x_i^2)}) - \exp(1/n \cdot \sum_i (\cos(2\pi \cdot x_i))) + 20 + e$ 
96.         x = np.array(x)
97.         sum_term = np.sum(x**2) / n
98.         cos_term = np.sum(np.cos(2 * np.pi * x)) / n
99.         return -20 * np.exp(-0.2 * np.sqrt(sum_term)) - np.exp(cos_term) + 20 + np.e
100.
101.     elif self.name == "griewank": #F11 - Multimodal Function
102.         #  $f(x) = 1 + 1/4000 \cdot \sum_i (x_i^2) - \prod_i (\cos(x_i/\sqrt{i}))$ 
103.         x = np.array(x)
104.         sum_term = np.sum(x**2) / 4000
105.         prod_term = np.prod(np.cos(x / np.sqrt(np.arange(1, n + 1))))
106.         return 1 + sum_term - prod_term
107.
108.     elif self.name == "rastrigin": #F9 - Multimodal Function
109.         #  $f(x) = 10n + \sum_i (x_i^2 - 10 \cdot \cos(2\pi \cdot x_i))$ 
110.         x = np.array(x)
111.         return 10 * n + np.sum(x**2 - 10 * np.cos(2 * np.pi * x))
112.
113.     elif self.name == "rosenbrock": #F5 - Unimodal Function
114.         #  $f(x) = \sum_i (100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2)$ 
115.         x = np.array(x)
116.         return np.sum(100 * (x[1:n] - (x[0 : n - 1]**2))**2 + (x[0 : n - 1] - 1)**2)
117.
118.     elif self.name == "sphere": #F1 - Unimodal Function
119.         #  $f(x) = \sum_i (x_i^2)$ 
120.         x = np.array(x)
121.         return np.sum(x**2)
122.
123.     elif self.name == "dejongF5_foxholes": #F14 - Fixed Dimension Multimodal
Function dimensions=2
124.         #  $f(x) = 1 / (0.002 + \sum_{i=1, 25} [1 / (i + (x_1 - A_{1i})^6 + (x_2 - A_{2i})^6)])$ 
125.         x1, x2 = x[0], x[1]
126.         A = np.zeros((2, 25))
127.         a = np.array([-32, -16, 0, 16, 32])
128.         A[0, :] = np.tile(a, 5)
129.         A[1, :] = np.repeat(a, 5)
130.
131.         sumterm1 = np.arange(1, 26)
132.         sumterm2 = (x1 - A[0, :])**6
133.         sumterm3 = (x2 - A[1, :])**6
134.
135.         sum_terms = np.sum(1 / (sumterm1 + sumterm2 + sumterm3))
136.         return 1 / (0.002 + sum_terms)
137.

```

```
138.     elif self.name == "goldstein-price": #F18 - Fixed Dimension Multimodal Function
dimensions=2
139.         #f(x1, x2) = (1 + ((x1 + x2 + 1)2)(19 - 14x1 + 3x12 - 14x2 + 6x1x2 + 3x22))
140.         #           × (30 + ((2x1 - 3x2)2)(18 - 32x1 + 12x12 + 48x2 - 36x1x2 + 27x22))
141.         x = np.array(x)
142.         x1 = x[0]
143.         x2 = x[1]
144.         subterm1a = (x1 + x2 + 1)**2
145.         subterm1b = 19 - 14*x1 + 3*x1**2 - 14*x2 + 6*x1*x2 + 3*x2**2
146.
147.         term1 = 1 + subterm1a * subterm1b
148.         subterm2a = (2*x1 - 3*x2)**2
149.         subterm2b = 18 - 32*x1 + 12*x1**2 + 48*x2 - 36*x1*x2 + 27*x2**2
150.
151.         term2 = 30 + subterm2a * subterm2b
152.         return term1 * term2
153.     else:
154.         raise ValueError("Unknown benchmark function")
```

A-5 Αρχείο plot_graph.py

```
1. # -*- coding: utf-8 -*-
2. """
3. @Student: Christos Smarlamakis
4. std147661@ac.eap.gr, christossmarlamakis@gmail.com / 6945830515
5. """
6.
7. import numpy as np
8. import matplotlib.pyplot as plt
9.
10. def plot_graph(benchmark_function, wolves_position_array, wolves_fitness,
iteration_counter):
11.     """
12.     Απεικονίζει την γραφική παράσταση της συνάρτησης αξιολόγησης και τις θέσεις των
λύκων κατα την σύγκλιση τους προς το βέλτιστο.
13.     """
14.     #Αποθήκευση του ολικού ελαχίστου στην μεταβλητή global_minimum.
15.     global_minimum = benchmark_function.get_fminimum_point()
16.     #Δημιουργία 3D Αξονα.
17.     ax = plt.axes(projection='3d')
18.     #Αποθήκευση σε κατάλληλες μεταβλητές των ορίων της συνάρτησης.
19.     lower_bound, upper_bound = benchmark_function.lower_bound,
benchmark_function.upper_bound
20.     #Δημιουργία πλέγματος των Αξόνων X1,X2.
21.     X1, X2 = np.linspace(lower_bound, upper_bound, 50), np.linspace(lower_bound,
upper_bound, 50)
22.     X, Y = np.meshgrid(X1, X2)
23.     #Υπολογισμός των τιμών της συνάρτησης για κάθε συντεταγμένη στο πλέγμα.
24.     Z = np.array([[benchmark_function.calculate_fitness([x, y]) for x, y in zip(row_x,
row_y)] for row_x, row_y in zip(X, Y)])
25.     #Σχεδίαση της συνάρτησης αξιολόγησης
26.     ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6)
27.     # Σχεδίαση των λύκων στον χώρο.
28.     ax.scatter3D(wolves_position_array[:, 0], wolves_position_array[:, 1],
wolves_fitness, color='red', marker='o', label="Search agent")
29.     # Σχεδίαση του ολικού ελάχιστου.
30.     plt.scatter(global_minimum[0], global_minimum[1], color='blue', s=100, marker='+',
label='Global Minimum')
31.     #Ρύθμιση της γωνίας προβολής του γραφήματος.
32.     ax.view_init(elev=30, azim=140)
33.     #Ορισμός Τίτλου και Ετικετών στους Αξονες.
34.     ax.set_title(f'{benchmark_function.name} function - Iteration {iteration_counter}')
35.     ax.set_xlabel('X1')
36.     ax.set_ylabel('X2')
37.     ax.set_zlabel('Fitness Value')
38.     plt.legend(loc='upper right', prop={'size': 8})
39.     # Ενημέρωση του γραφήματος σε κάθε επανάληψη
```

```

40. plt.pause(0.001)
41.
42.
43.     def plot_top_view_graph(benchmark_function, wolves_position_array,
iteration_counter):
44.         """
45.         Απεικονίζει την κάτοψη των λύκων και τη σύγκλισή τους προς το ολικό βέλτιστο.
46.         """
47.         #Αποθήκευση σε κατάλληλες μεταβλητές των ορίων και του ολικού βέλτιστου της
συνάρτησης.
48.         lower_bound, upper_bound = benchmark_function.lower_bound,
benchmark_function.upper_bound
49.         global_minimum = benchmark_function.get_fminimum_point()
50.         # Δημιουργία γραφήματος με μέγεθος 10x6.
51.         plt.figure(figsize=(10, 6))
52.         # Δημιουργία πλέγματος για τον άξονα X1 και X2.
53.         X1, X2 = np.linspace(lower_bound, upper_bound, 50), np.linspace(lower_bound,
upper_bound, 50)
54.         X, Y = np.meshgrid(X1, X2)
55.         # Υπολογισμός τιμών της συνάρτησης για κάθε συντεταγμένη πάνω στο πλέγμα.
56.         Z = np.array([[benchmark_function.calculate_fitness([x, y]) for x, y in zip(row_x,
row_y)] for row_x, row_y in zip(X, Y)])
57.         plt.contourf(X, Y, Z, levels=50, cmap='viridis', alpha=0.6)
58.         plt.colorbar(label='Fitness Value')
59.         # Σχεδίαση των θέσεων των λύκων
60.         plt.scatter(wolves_position_array[:, 0], wolves_position_array[:, 1], color='red', s=50,
label='Search agent')
61.         # Σχεδίαση του ολικού ελάχιστου
62.         plt.scatter(global_minimum[0], global_minimum[1], color='blue', s=100, marker='+',
label='Global Minimum')
63.         plt.title(f'{benchmark_function.name} function Top View- Iteration
{iteration_counter}')
64.         plt.xlabel('X1')
65.         plt.ylabel('X2')
66.         plt.legend(loc='upper right', prop={'size': 8})
67.         # Ενημέρωση του γραφήματος σε κάθε επανάληψη.
68.         plt.pause(0.001)

```

Παράρτημα Β «Χαρακτηριστικά Υπολογιστικού Συστήματος Δοκιμών»

Κατά την εκτέλεση των πειραμάτων χρησιμοποιήθηκε το παρακάτω υπολογιστικό σύστημα:

Processors Information	
Socket 1	ID = 0
Number of cores	4 (max 4)
Number of threads	4 (max 4)
Manufacturer	GenuineIntel
Name	Intel Core i5 6500T
Codename	Skylake
Specification	Intel(R) Core(TM) i5-6500T @ 2.50GHz
Package (platform ID)	Socket 1151 LGA (0x1)
CPUID	6.E.3
Extended CPUID	6.5E
Core Stepping	R0
Technology	14 nm
TDP Limit	35.0 Watts
Tjmax	100.0 °C
Core Speed	2791.8 MHz
Multiplier x Bus Speed	28.0 x 99.7 MHz
Base frequency (cores)	99.7 MHz
Stock frequency	2500 MHz
Max frequency	3100 MHz
Instructions sets	MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, EM64T, VT-x, AES, AVX, AVX2, FMA3, TSX
Microcode Revision	0xF0
L1 Data cache	4 x 32 KB (8-way, 64-byte line)
L1 Instruction cache	4 x 32 KB (8-way, 64-byte line)
L2 cache	4 x 256 KB (4-way, 64-byte line)
L3 cache	6 MB (12-way, 64-byte line)
Max CPUID level	00000016h
Max CPUID ext. level	80000008h
FID/VID Control	yes

Πίνακας Β - 1 : Τεχνικά Χαρακτηριστικά CPU

Chipset

Northbridge	Intel Skylake rev. 07
Southbridge	Intel Q170 rev. 31
Bus Specification	PCI-Express 3.0 (8.0 GT/s)
Graphic Interface	PCI-Express
Memory Type	DDR4
Memory Size	16 GBytes
Channels	Single
Memory Frequency	1063.5 MHz (3:32)
CAS# latency (CL)	15.0
RAS# to CAS# delay (tRCD)	15
RAS# Precharge (tRP)	15
Cycle Time (tRAS)	35
Row Refresh Cycle Time (tRFC)	374
Command Rate (CR)	2T
Uncore Frequency	2793.2 MHz
Host Bridge	0x191F

Πίνακας Β - 2 : Τεχνικά Χαρακτηριστικά RAM

Display Adapters

Display adapter 0 (primary)	
ID	0x4000000
Name	Intel(R) HD Graphics 530
Board Manufacturer	Hewlett-Packard
PCI device	bus 0 (0x0), device 2 (0x2), function 0 (0x0)
Vendor ID	0x8086 (0x103C)
Model ID	0x1912 (0x8055)
Revision ID	0x6
Performance Level	Current
Core clock	498.7 MHz
Driver version	31.0.101.2111
WDDM Model	2.1
Display Adapters	
Display adapter 0 (primary)	
ID	0x4000000
Name	Intel(R) HD Graphics 530

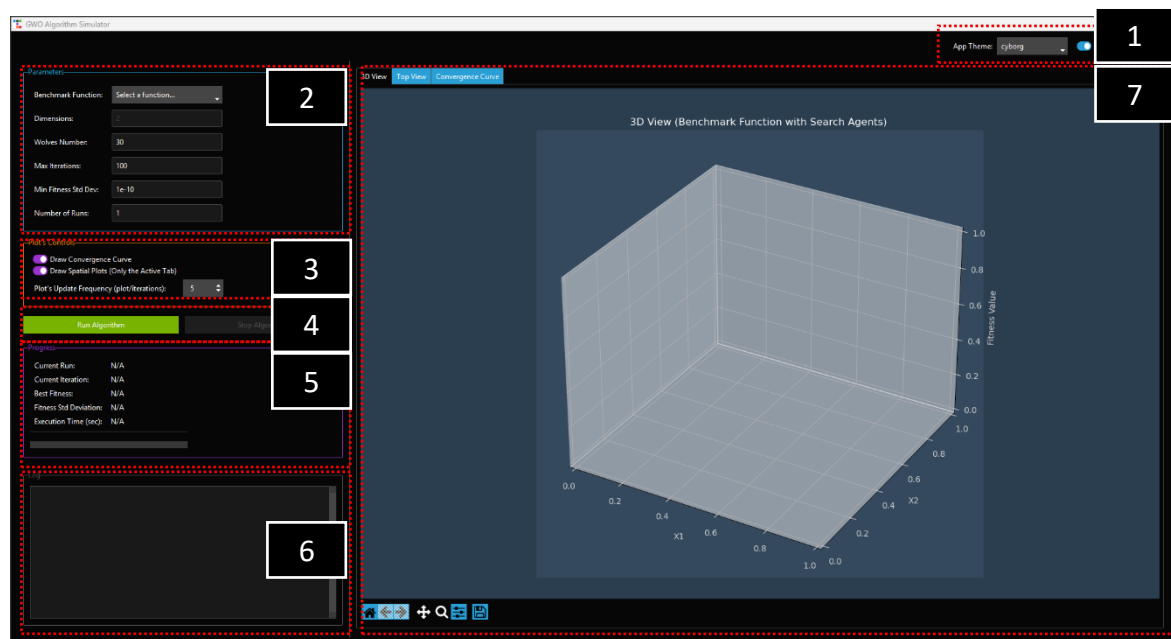
Πίνακας Β - 3 : Τεχνικά Χαρακτηριστικά GPU

Παράρτημα Γ «Γραφικό Περιβάλλον Διεπαφής (GUI)»

Γ-1 Εισαγωγή

Για σκοπούς οπτικοποίησης και ευχερέστερης παραμετροποίησης του αλγορίθμου αναπτύχθηκε ένα Γραφικό Περιβάλλον Διεπαφής (GUI) με χρήση της βιβλιοθήκης Tkinter και της επέκτασης της ttkbootstrap για βελτιωμένη αισθητική. Η οπτικοποίηση των αποτελεσμάτων και της διαδικασίας σύγκλισης υλοποιήθηκε με την ενσωμάτωση της βιβλιοθήκης Matplotlib. Το GUI επιτρέπει στον χρήστη να εισάγει εύκολα τις παραμέτρους του αλγορίθμου, να παρακολουθεί την πρόοδο του σε πραγματικό χρόνο και να εξάγει τα αποτελέσματα.

Γ-2 Κύρια Οθόνη και Λειτουργίες



Εικόνα Γ - 1 : Κύρια Οθόνη GWO Simulator

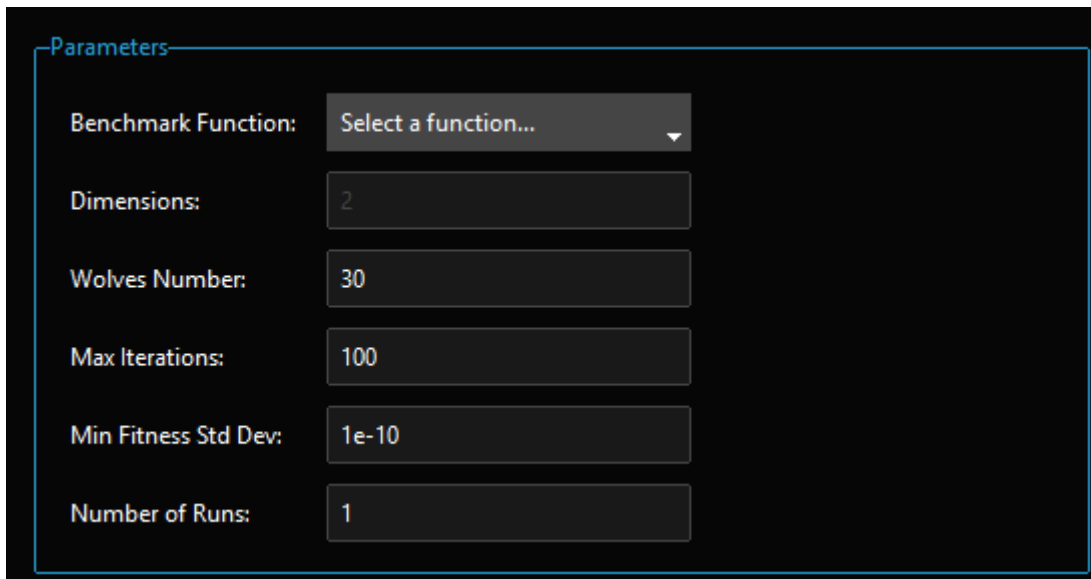
Τα βασικά στοιχεία της οθόνης είναι τα εξής:

1. Πλαίσιο Επιλογής Θέματος (Πάνω Δεξιά). Ο χρήστης πραγματοποιεί την επιλογή του θέματος της εφαρμογής, παρέχονται φωτεινά και σκουρόχρωμα θέματα ενώ υφίσταται και κουμπί για την επιλογή φωτεινού ή σκούρου φόντου στις απεικονίσεις των γραφημάτων ανεξάρτητα από το θέμα της εφαρμογής.
2. Πλαίσιο Εισαγωγής Παραμέτρων (Αριστερά). Ο χρήστης επιλέγει την συνάρτηση αξιολόγησης, τις διαστάσεις του προβλήματος, τον αριθμό των πρακτόρων αναζήτησης

- (λύκων), τον μέγιστο αριθμό επαναλήψεων του αλγορίθμου, την ελάχιστη τυπική απόκλιση καταλληλοτήτων των λύκων για τον τερματισμό του αλγορίθμου και τον αριθμό των εκτελέσεων.
3. Πλαίσιο Ελέγχου Γραφημάτων (Αριστερά, κάτω από το πλαίσιο των παραμέτρων). Σε αυτό το πλαίσιο υπάρχουν κουμπιά ελέγχου για την ενεργοποίηση της σχεδίασης των γραφημάτων της σύγκλισης της καταλληλότητας και των χωρικών γραφημάτων που απεικονίζουν την γραφική παράσταση της συνάρτησης αξιολόγησης και τις θέσεις των πρακτόρων αναζήτησης. Επίσης επιλέγουμε την συχνότητα με την οποία επιθυμούμε να γίνεται η σχεδίαση των γραφημάτων.
 4. Πλαίσιο Κουμπιών Ελέγχου Εκτέλεσης (Αριστερά, κάτω από το προηγούμενο πλαίσιο). Σε αυτό το σημείο βρίσκονται τα κουμπιά “Run Algorithm” και “Stop Algorithm” για την έναρξη και παύση της εκτέλεσης του αλγορίθμου.
 5. Πλαίσιο Προόδου (Αριστερά, κάτω από τα κουμπιά έναρξης/παύσης). Εμφανίζονται πληροφορίες για την τρέχουσα εκτέλεση (RUN) , την τρέχουσα επανάληψη, την βέλτιστη τιμή καταλληλότητας που έχει υπολογιστεί, την τυπική απόκλιση των καταλληλοτήτων των λύκων και τον χρόνο εκτέλεσης. Περιλαμβάνεται επίσης και μία μπάρα προόδου για την οπτική απεικόνιση της προόδου ολοκλήρωσης της αναζήτησης στον χώρο των λύσεων.
 6. Περιοχή Καταγραφής – Log (Κάτω Αριστερή Γωνία). Σε αυτή την περιοχή εμφανίζονται μηνύματα προς τον χρήστη για την ενημέρωση του για διάφορες ενέργειες που λαμβάνουν χώρα στο πλαίσιο λειτουργίας της εφαρμογής καθώς της εκτέλεσης του αλγορίθμου, όπως είναι τυχόν σφάλματα και της αποθήκευσης των αρχείων αποτελεσμάτων της εκτέλεσης.
 7. Περιοχή Γραφημάτων (Δεξιά). Σε αυτή την περιοχή περιέχονται καρτέλες, η κάθε καρτέλα φιλοξενεί το κάθε γράφημα αναλόγως του είδους του. (3D / Top View, Convergence Curve).

Γ-3 Οδηγίες Χρήσης

- Εισαγωγή Παραμέτρων



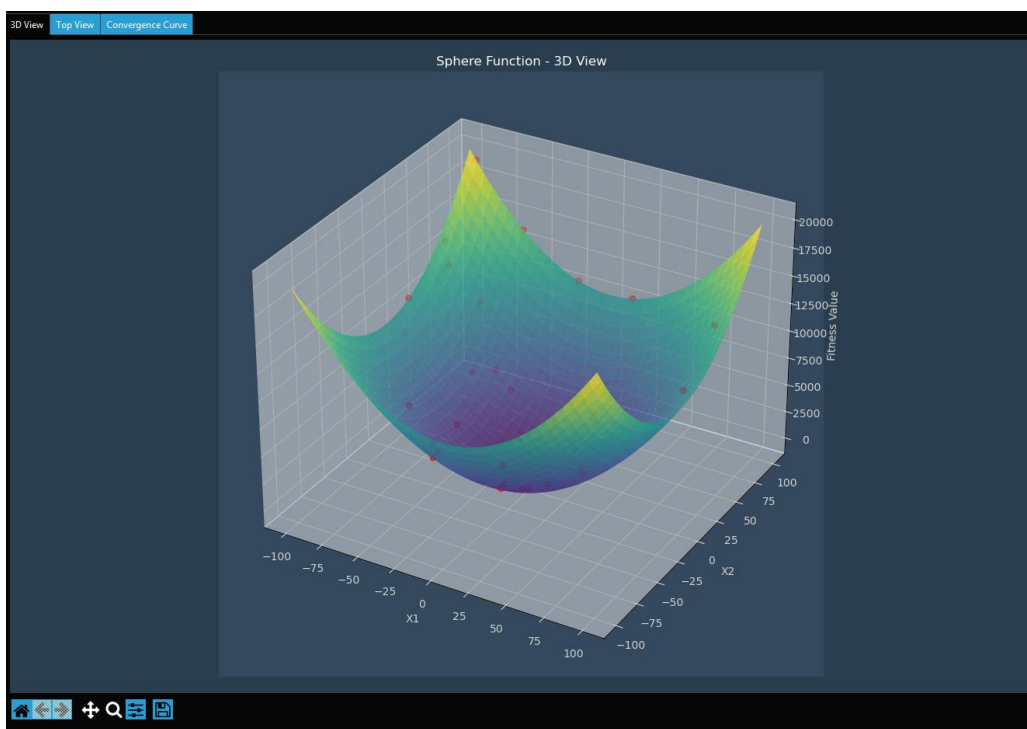
Εικόνα Γ - 2 : Πλαίσιο Εισαγωγής Παραμέτρων

Ο χρήστης επιλέγει αρχικά την επιθυμητή συνάρτηση αξιολόγησης από την αναδιπλούμενη λίστα. Για τις συναρτήσεις πολλών διαστάσεων (π.χ Ackley, Sphere) ο χρήστης μπορεί να ορίσει τον επιθυμητό αριθμό διαστάσεων. Στις συναρτήσεις σταθερών διαστάσεων (π.χ Goldstein-Price) το πεδίο απενεργοποιείται αυτόματα και τίθεται σε 2. Ακολούθως ο χρήστης επιλέγει όπως προαναφέρθηκε τις υπόλοιπες παραμέτρους.

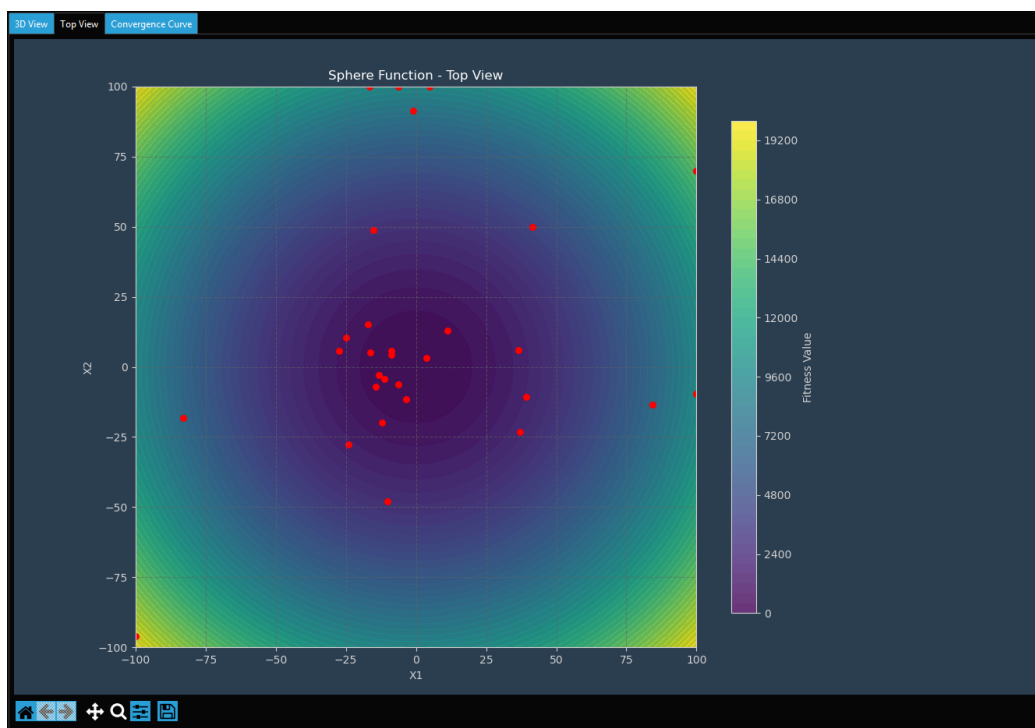
- Έναρξη και Παρακολούθηση Εκτέλεσης

Πατώντας το κουμπί “Run Algorithm” ξεκινά η εκτέλεση του GWO. Τα κουμπιά ελέγχου ενημερώνονται με αποτέλεσμα το κουμπί “Run” να απενεργοποιηθεί και το κουμπί “Stop Algorithm” να ενεργοποιηθεί. Επίσης το πλαίσιο με την εμφάνιση της προόδου ενημερώνεται και τα αντίστοιχα δεδομένα εμφανίζονται προς ενημέρωση του χρήστη. Στην περιοχή Log εμφανίζονται διάφορα μηνύματα σχετικά με την πρόοδο εκτέλεσης.

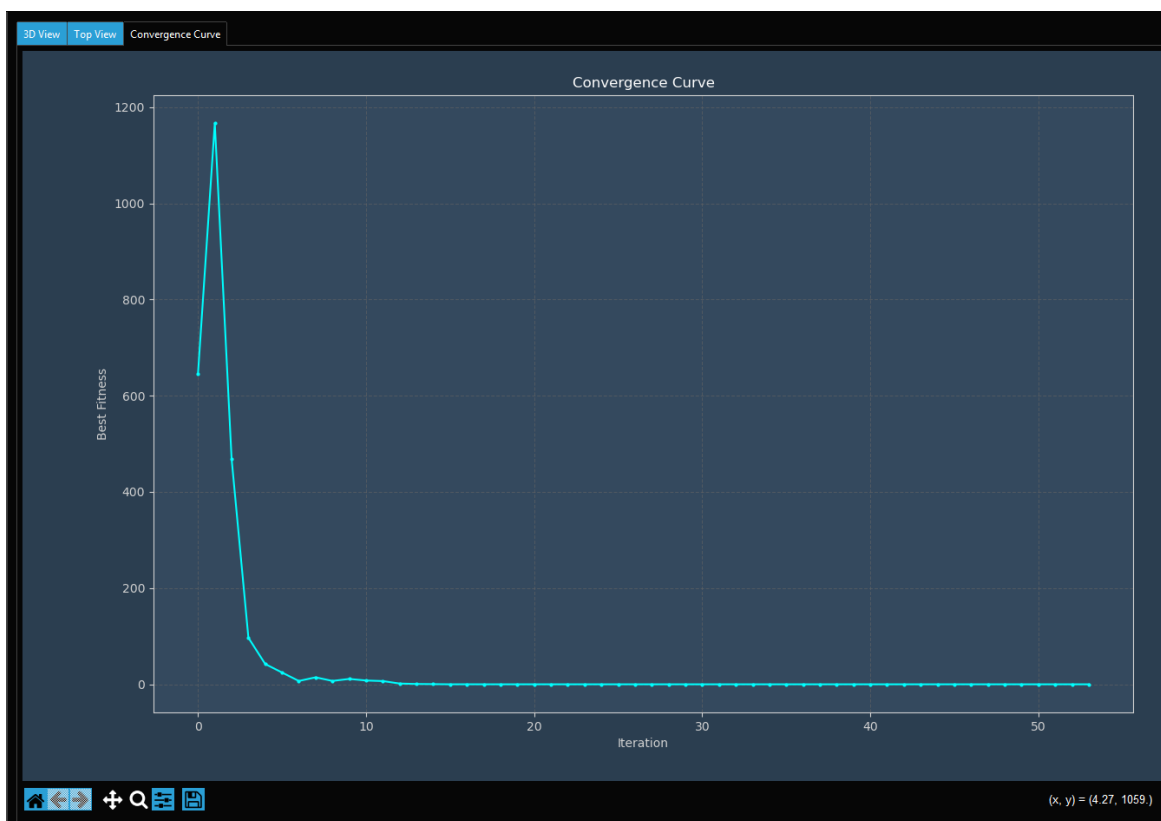
- **Οπτικοποίηση Αποτελεσμάτων**



Εικόνα Γ - 3 : Παράδειγμα Γραφήματος 3D View της Διαδικασίας Βελτιστοποίησης



Εικόνα Γ - 4 : Παράδειγμα Γραφήματος Top View



Εικόνα Γ - 5 : Παράδειγμα Καμπύλης Σύγκλισης

Κατά την διάρκεια εκτέλεσης του αλγορίθμου αν έχουν ενεργοποιηθεί τα κουμπιά για την απεικόνιση των γραφημάτων τότε τα γραφήματα απεικονίζονται στις αντίστοιχες καρτέλες με την διαφοροποίηση ότι για τα χωρικά γραφήματα θα πρέπει να είναι ενεργοποιημένη αι η αντίστοιχη καρτέλα. Στο κάτω μέρος κάθε γραφήματος διατίθεται η γραμμή εργαλείων της Matplotlib για διάδραση του χρήστη (Zoom, Pan, Save).

- **Αλλαγή Θέματος και Φόντου Γραφημάτων**

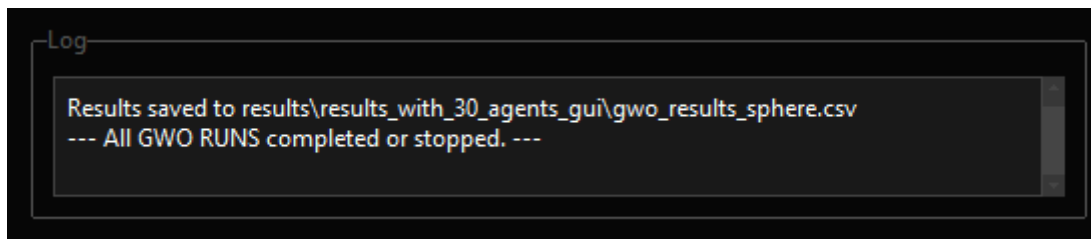


Εικόνα Γ - 6 : Επιλογές Θέματος και Φόντου Γραφημάτων

Ο Χρήστης μπορεί να αλλάξει το θέμα της εφαρμογής από την αναδυόμενη λίστα “App Theme”. Επιπροσθέτως με το κουμπί “Dark Plot Background” μπορεί όπως προαναφέρθηκε να επιλέξει μεταξύ ανοιχτόχρωμου και σκούρου φόντου γραφημάτων για μεγαλύτερη οπτική άνεση.

- **Έξοδος Αποτελεσμάτων**

Με την ολοκλήρωση όλων των εκτελέσεων (runs) ή με την διακοπή από τον χρήστη, τα συγκεντρωτικά αποτελέσματα κάθε εκτέλεσης (run) αποθηκεύονται αυτόματα σε ένα αρχείο CSV. Το όνομα του αρχείου (π.χ gwo_results_sphere.csv) καθώς και η διαδρομή αποθήκευσης του (στον υποφάκελο results/results_with_X_agents_gui/) εμφανίζονται στην περιοχή καταγραφής (Log).



Εικόνα Γ - 7 : Απεικόνιση Πληροφοριών στην περιοχή Log

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.