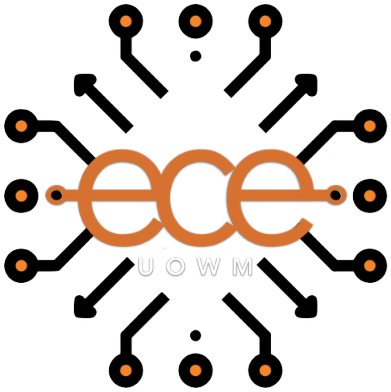


Πανεπιστήμιο Δυτικής Μακεδονίας

Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών

Αυτοματοποιημένη εύρεση βέλτιστων παραμέτρων λογισμικού

Δημήτριος Ζούνος (ΑΜ: 1329)

Επιβλέπων Καθηγητής: Νικόλαος Πλόσκας

Εργαστήριο Ευφυών Συστημάτων & Βελτιστοποίησης

18 Οκτωβρίου 2022

Περίληψη

Οι υπερπαράμετροι είναι παράμετροι που δε μεταβάλλονται και επιλέγονται πριν την εκτέλεση ενός αλγόριθμου ή λογισμικού. Η επιλογή υπερπαραμέτρων είναι σημαντική γιατί μπορεί να επηρεάσει την απόδοση του λογισμικού. Για παράδειγμα κατά την εκπαίδευση ενός μοντέλου τεχνητής νοημοσύνης, η επιλογή βέλτιστων υπερπαραμέτρων μπορεί να οδηγήσει σε αύξηση της απόδοσης του μοντέλου ενώ ή κακή επιλογή μπορεί να μειώσει την απόδοση δραματικά. Τέτοιου είδους προβλήματα μπορούν να θεωρηθούν και ως προβλήματα μαύρου κουτιού, καθώς συμπεράσματα μπορούμε να εξάγουμε μόνο από τα αποτελέσματα χωρίς να γνωρίζουμε τη δομή του προβλήματος. Η επιλογή των βέλτιστων υπερπαραμέτρων δεν είναι πάντα μια εύκολη διαδικασία. Κάθε πρόβλημα και κάθε λογισμικό είναι μοναδικό και περίπλοκο με το δικό του τρόπο και έτσι απαιτούνται γνώσεις πάνω στο αντικείμενο για τη βέλτιστη επιλογή των υπερπαραμέτρων του. Η ανάγκη για εργαλεία που μπορούν να κάνουν αυτή τη διαδικασία αυτοματοποιημένα και να επιλέξουν τις κατάλληλες υπερπαραμέτρους για κάθε περίπτωση και να δώσουν πληροφορίες και καθοδήγηση στους χρήστες για το πως λύνεται το πρόβλημα αυξάνεται διαρκώς. Σε αυτή τη διπλωματική εργασία παρουσιάζονται πειραματικά αποτελέσματα για την εξαγωγή συμπερασμάτων για τη βελτιστοποίηση του χρόνου επίλυσης μαθηματικών προβλημάτων βελτιστοποίησης με την εύρεση βέλτιστων υπερπαραμέτρων του λύτη Gurobi με τη χρήση αλγορίθμων που υλοποιήθηκαν και τη χρήση συστημάτων λογισμικού που είναι εξειδικευμένα για τη λύση του προβλήματος αυτού.

Λέξεις κλειδιά: Υπερπαράμετροι, βελτιστοποίηση, προβλήματα μαύρου κουτιού, μαθηματικός προγραμματισμός

Abstract

Hyperparameters are parameters that do not change and are selected before an algorithm or software is executed. The selection of hyperparameters is important because it can affect the performance of the software. For example, when training an artificial intelligence model, choosing optimal hyperparameters can lead to an increase in the performance of the model while poor selection can decrease the performance dramatically. Such problems can also be considered as black box problems, as conclusions can only be drawn from the results without knowing the structure of the problem. Choosing the optimal hyperparameters is not always an easy process. Each problem and each software is unique and complex in its own way and thus knowledge about the domain is required to optimally select its hyperparameters. The need for tools that can do this process automatically and select the appropriate hyperparameters for each case and provide information and guidance to users on how to solve the problem is constantly growing. In this thesis, experimental results are presented to draw conclusions about the optimization of the time to solve mathematical optimization problems by finding optimal hyperparameters of the Gurobi solver using algorithms implemented and the use of software systems specialized for solving this problem.

Keywords: Hyperparameters, optimization, black box problems, mathematical programming, Gurobi

Δήλωση Πνευματικών Δικαιωμάτων

Δήλωση Πνευματικών Δικαιωμάτων Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν. 1599/1986 και τα άρθρα 2,4,6 παρ. 3 του Ν. 1256/1982, η παρούσα Διπλωματική Εργασία με τίτλο ''Αυτοματοποιημένη εύρεση βέλτιστων παραμέτρων λογισμικού'' καθώς και τα ηλεκτρονικά αρχεία και πηγαίοι κώδικες που αναπτύχθηκαν ή τροποποιήθηκαν στα πλαίσια αυτής της εργασίας και αναφέρονται ρητώς μέσα στο κείμενο που συνοδεύουν, και η οποία έχει εκπονηθεί στο Τμήμα Ηλεκτρολόγων Μηχανικών & Μηχανικών Υπολογιστών του Πανεπιστημίου Δυτικής Μακεδονίας, υπό την επίβλεψη του μέλους του Τμήματος κ. Νικολάου Πλόσκα, αποτελεί αποκλειστικά προϊόν προσωπικής εργασίας και δεν προσβάλλει κάθε μορφής πνευματικά δικαιώματα τρίτων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον. Τα σημεία όπου έχω χρησιμοποιήσει ιδέες, κείμενο, αρχεία ή / και πηγές άλλων συγγραφέων, αναφέρονται ευδιάκριτα στο κείμενο με την κατάλληλη παραπομπή και η σχετική αναφορά περιλαμβάνεται στο τμήμα των βιβλιογραφικών αναφορών με πλήρη περιγραφή.

Απαγορεύεται η αντιγραφή, αποθήκευση και διανομή της παρούσας εργασίας, εξ ολοκλήρου ή τμήματος αυτής, για εμπορικό σκοπό. Επιτρέπεται η ανατύπωση, αποθήκευση και διανομή για σκοπό μη κερδοσκοπικό, εκπαιδευτικής ή ερευνητικής φύσης, υπό την προϋπόθεση να αναφέρεται η πηγή προέλευσης και να διατηρείται το παρόν μήνυμα. Ερωτήματα που αφορούν τη χρήση της εργασίας για κερδοσκοπικό σκοπό πρέπει να απευθύνονται προς τον συγγραφέα. Οι απόψεις και τα συμπεράσματα που περιέχονται σε αυτό το έγγραφο εκφράζουν τον συγγραφέα και μόνο.

Υπογραφή Φοιτητή

Περιεχόμενα

1 Εισαγωγή **10**

1.1 Ορισμός του προβλήματος 10

1.2 Κίνητρα και στόχοι υλοποίησης 11

1.3 Διάρθρωση κειμένου 11

2 Βελτιστοποίηση υπερπαραμέτρων **12**

2.1 Βελτιστοποίηση 12

2.2 Υπερπαράμετροι (Hyperparameters) 13

2.3 Πρόβλημα εύρεσης βέλτιστων υπερπαραμέτρων 14

2.4 Μη αυτοματοποιημένη αναζήτηση βέλτιστων υπερπαραμέτρων 15

2.5 Αυτοματοποιημένη αναζήτηση βέλτιστων υπερπαραμέτρων 15

2.5.1 Αλγόριθμοι δειγματοληψίας (Sampling algorithms) 16

2.5.2 Μπεϋζιανή βελτιστοποίηση (Bayesian optimization) 17

2.5.3 Εξελικτική υπολογιστική 18

2.5.4 Βελτιστοποίηση χωρίς παράγωγο 22

3 Υλοποίηση **24**

3.1 Απλό παράδειγμα βελτιστοποίησης 24

3.2 Αλγόριθμος αναζήτησης πλέγματος (Grid search) 25

3.2.1 Παράδειγμα εκτέλεσης αλγορίθμου 29

3.3 Αλγόριθμος τυχαίας αναζήτησης (Random search) 30

3.3.1 Παράδειγμα εκτέλεσης αλγορίθμου 32

3.4 Ακολουθίες χαμηλής απόκλισης 33

3.4.1 Ακολουθία Van der Corput 34

3.4.2 Ακολουθία Halton 34

3.4.3 Ακολουθία Hammersly 39

3.4.4 Ακολουθία Sobol 43
3.5 Λατινικός υπερκύβος (Latin hyperCube) 50
3.5.1 Παράδειγμα αλγορίθμου LHS 53
3.6 Συστήματα λογισμικού εύρεσης βέλτιστων υπερπαραμέτρων 54
3.6.1 Βασικά στοιχεία του αλγορίθμου MADS 54
3.6.2 Παράδειγμα εκτέλεσης με το λογισμικό Dakota 57
3.6.3 Παράδειγμα εκτέλεσης με το λογισμικό Nomad 58

4 Υπολογιστική μελέτη **60**
4.1 Πρόγραμμα μαύρο κουτί 60
4.1.1 Λύτης γραμμικού προγραμματισμού Gurobi 61
4.1.2 Διαδικασία εκτέλεσης της μελέτης 63
4.2 Αποτελέσματα 68

5 Συμπεράσματα **80**

Κατάλογος σχημάτων

2.1 Μια γενική εικόνα τρόπου λειτουργίας για έναν απλό εξελικτικό αλγόριθμο 20

3.1 Περιγραφή προβλήματος ως ένα μαύρο κουτί 25
3.2 Συνδυασμοί παραμέτρων αλγορίθμου αναζήτησης πλέγματος 29
3.3 Συνδυασμοί παραμέτρων αλγορίθμου τυχαίας αναζήτησης 32
3.4 Συνδυασμοί παραμέτρων αλγορίθμου Halton 38
3.5 Συνδυασμοί παραμέτρων αλγορίθμου Hammersly 42
3.6 Συνδυασμοί παραμέτρων αλγορίθμου Sobol 49
3.7 Συνδυασμοί παραμέτρων αλγορίθμου λατινικού υπερκύβου 53
3.8 Συνδυασμοί παραμέτρων λογισμικού Dakota MADS 57
3.9 Συνδυασμοί παραμέτρων λογισμικού Nomad 58

4.1 Ποσοστό βελτίωσης συνολικού χρόνου όλων των προβλημάτων 69
4.2 Ποσοστό λυμένων προβλημάτων σε σχέση με το χρόνο 70
4.3 Μέσος όρος βελτίωσης χρόνου όλων των προβλημάτων 71
4.4 Μέσος όρος εύρεσης καλύτερων αποτελεσμάτων στις 100 δοκιμές ανά πρόβλημα 72
4.5 Μέσος όρος θέσης που βρέθηκε η καλύτερη λύση 73
4.6 Μέσος όρος χρόνου εκτέλεσης αλγορίθμων 74
4.7 Ποσοστό προβλημάτων με καλύτερο χρόνο ανά αλγόριθμο 75

Κατάλογος αλγορίθμων

1 Βελτιστοποίηση υπερπαραμέτρων αλγορίθμου αναζήτησης πλέγματος 27
2 Βελτιστοποίηση υπερπαραμέτρων αλγορίθμου τυχαίας αναζήτησης . . 31
3 Βελτιστοποίηση υπερπαραμέτρων με την ακολουθία Van der Corput . 34
4 Βελτιστοποίηση υπερπαραμέτρων με την ακολουθία Halton 36
5 Βελτιστοποίηση υπερπαραμέτρων με την ακολουθία Hammersly . . . 40
6 Βελτιστοποίηση υπερπαραμέτρων με την ακολουθία Sobol 47
7 Βελτιστοποίηση υπερπαραμέτρων αλγορίθμου λατινικού υπερκύβου . 52
8 Βελτιστοποίηση υπερπαραμέτρων με τον αλγόριθμο MADS 56
9 Εκτελέσιμο αρχείο 61

Κατάλογος πινάκων

4.1 Υπερπαράμετροι λύτη Gurobi 62
4.2 Τιμές υπερπαραμέτρων λύτη Gurobi 63
4.3 Πληροφορίες προβλημάτων 64
4.4 Αποτελέσματα βελτιστοποίησης 76

Κατάλογος απεικονίσεων

3.1 Παράδειγμα αλγορίθμου αναζήτησης πλέγματος 29
3.2 Παράδειγμα αλγορίθμου τυχαίας αναζήτησης 32
3.3 Παράδειγμα αλγορίθμου Halton 38
3.4 Παράδειγμα αλγορίθμου Hammersly 42
3.5 Παράδειγμα εκτέλεσης του αλγορίθμου Sobol 49
3.6 Παράδειγμα αλγορίθμου Λατινικός υπερκύβος 53
3.7 Παράδειγμα λογισμικού Dakota με MADS 57
3.8 Παράδειγμα λογισμικού Nomad με MADS 58

Κεφάλαιο 1

Εισαγωγή

1.1 Ορισμός του προβλήματος

Οι αλγόριθμοι και τα συστήματα λογισμικού συνήθως αποτελούνται από πολλές παραμέτρους που επιλέγονται στην αρχή και δε μεταβάλλονται σε όλη τη διάρκεια της εκτέλεσης τους. Αυτού του τύπου οι παράμετροι ονομάζονται υπερπαράμετροι και επηρεάζουν άμεσα την απόδοση των αλγορίθμων και των συστημάτων λογισμικού. Για να εκτελεστούν αποτελεσματικά θα πρέπει να βελτιστοποιηθούν οι υπερπαράμετροι τους. Η βελτιστοποίηση υπερπαραμέτρων λειτουργεί εκτελώντας πολλαπλές δοκιμές με την κάθε δοκιμή να είναι μια πλήρης εκτέλεση του αλγορίθμου ή του λογισμικού με τιμές για τις υπερπαραμέτρους που έχουν επιλεχθεί.

Μπορεί να είναι δυνατό να βελτιστοποιηθούν μερικές υπερπαράμετροι με μη αυτόματο τρόπο για έναν απλό αλγόριθμο ή λογισμικό, ωστόσο, καθώς ο αλγόριθμος ή το λογισμικό γίνεται πιο σύνθετος, ο αριθμός των υπερπαραμέτρων αυξάνεται επίσης, γεγονός που κάνει τη βελτιστοποίηση υπερπαραμέτρων περισσότερο δύσκολη. Ως αποτέλεσμα, η αυτοματοποίηση της βελτιστοποίησης υπερπαραμέτρων θα είχε μεγάλο ενδιαφέρον σε πολλές διαφορετικές εφαρμογές μειώνοντας τη χειρωνακτική εργασία και αυξάνοντας παράλληλα την απόδοση του αλγορίθμου ή του λογισμικού. Στην παρούσα διπλωματική εργασία επιτυγχάνουμε με τη χρήση συστημάτων λογισμικού και αλγορίθμων να βρεθούν υπερπαράμετροι τέτοιοι ώστε ο λύτης γραμμικού προγραμματισμού Gurobi να λύσει σε λιγότερο χρόνο επιλεγμένα προβλήματα σε σχέση με τις προεπιλεγμένες τιμές του λογισμικού.

1.2 Κίνητρα και στόχοι υλοποίησης

Η εύρεση βέλτιστων υπερπαραμέτρων μπορεί να μετατραπεί σε ένα πολύ δύσκολο και χρονοβόρο πρόβλημα ειδικά όταν είναι πολυδιάστατο, δηλαδή όταν εμπλέκονται πολλοί υπερπαράμετροι που χρειάζεται να βελτιστοποιηθούν. Η χειροκίνητη επιλογή των βέλτιστων υπερπαραμέτρων πολλές φορές χρειάζεται μεγάλη εμπειρία από τον χρήστη και επομένως είναι ένα εγχείρημα που δύσκολα μπορεί να επαναληφθεί από μη έμπειρους χρήστες.

Η αυτοματοποίηση αυτής της διαδικασίας μπορεί να λύσει αυτό το πρόβλημα και να κάνει τη διαδικασία πιο εύκολη και προσβάσιμη σε όλους, καθώς και να επιφέρει και καλύτερα αποτελέσματα. Υπάρχουν διάφορα συστήματα λογισμικού και αλγόριθμοι για την αυτοματοποίηση αυτής της διαδικασίας. Στην παρούσα διπλωματική εργασία παρουσιάζονται διάφοροι αλγόριθμοι και συστήματα λογισμικού για την αυτοματοποιημένη εύρεση βέλτιστων υπερπαραμέτρων για τον λύτη γραμμικού προγραμματισμού Gurobi. Η ανάλυση των προβλημάτων μαθηματικού προγραμματισμού γίνεται ως προς την ελαχιστοποίηση του χρόνο επίλυσής τους.

1.3 Διάρθρωση κειμένου

Τα υπόλοιπα κεφάλαια είναι ως εξής: Στο κεφάλαιο 2 παρουσιάζεται το πρόβλημα εύρεσης βέλτιστων υπερπαραμέτρων και μέθοδοι που χρησιμοποιούνται για την αυτοματοποίηση του. Στο κεφάλαιο 3 παρουσιάζονται οι αλγόριθμοι και τα συστήματα λογισμικού που χρησιμοποιήθηκαν και επίσης πραγματοποιείται αυτοματοποιημένη εύρεση βέλτιστων υπερπαραμέτρων με τη χρήση ενός πολύ απλού παραδείγματος. Στο κεφάλαιο 4 πραγματοποιείται η υπολογιστική μελέτη και παρουσιάζονται τα αποτελέσματα που προέκυψαν από αυτή. Τέλος, στο κεφάλαιο 5 παρουσιάζονται τα συμπεράσματα της παρούσας εργασίας.

Κεφάλαιο 2

Βελτιστοποίηση υπερπαραμέτρων

2.1 Βελτιστοποίηση

Πολλά σημαντικά εφαρμοσμένα προβλήματα περιλαμβάνουν την εύρεση της καλύτερης λύσης. Συχνά αυτό περιλαμβάνει την εύρεση της μέγιστης ή ελάχιστης τιμής κάποιας συνάρτησης: ο ελάχιστος χρόνος για να πραγματοποιηθεί ένα συγκεκριμένο ταξίδι, το ελάχιστο κόστος για την εκτέλεση μιας διεργασίας, η μέγιστη ισχύς που μπορεί να παραχθεί από μια συσκευή και ούτω καθεξής. Η επίλυση προβλημάτων βελτιστοποίησης, δηλαδή η ελαχιστοποίηση ή η μεγιστοποίηση μιας αντικειμενικής συνάρτησης που περιλαμβάνει άγνωστες παραμέτρους/μεταβλητές, στις οποίες οι μεταβλητές μπορούν να είναι περιορισμένες σε ένα εύρος τιμών, ή ο μαθηματικός προγραμματισμός είναι ένα από τα βασικά στοιχεία των υπολογιστικών μαθηματικών[1]. Στο βιβλίο του για τα Εφαρμοσμένα Μαθηματικά, ο διαπρεπής μαθηματικός Gil Strang πιστεύει ότι η βελτιστοποίηση, μαζί με τη λύση συστημάτων γραμμικών εξισώσεων και διαφορικών εξισώσεων, είναι ένας από τους τρεις ακρογωνιαίους λίθους των σύγχρονων εφαρμοσμένων μαθηματικών[2].

Τα προβλήματα βελτιστοποίησης μπορούν γενικά να περιγραφούν είτε ως συνεχή είτε ως διακριτά, αλλά μπορεί να είναι ένας συνδυασμός και των δύο. Η διακριτή βελτιστοποίηση, όπως υποδηλώνει το όνομά της, αφορά την περίπτωση όπου οι μεταβλητές μπορούν να λάβουν μόνο διακριτές (και συνήθως ακέραιες) τιμές. Συχνά αυτά τα προβλήματα είναι πολύ δύσκολα και μόνο η απαρίθμηση όλων των πιθανών σημείων εγγυάται την καλύτερη λύση. Ευτυχώς, μερικές φορές τα προβλήματα είναι εύκολα και οι απλές (άπληστες) ευρετικές μέθοδοι εγγυούνται τη λύση τους. Αντίθετα, οι μεταβλητές στα προβλήματα συνεχούς βελτιστοποίησης επιτρέπεται να

λαμβάνουν οποιεσδήποτε τιμές που επιτρέπονται εντός των περιορισμών των μεταβλητών. Η διαδικασία της βελτιστοποίησης πολλές φορές είναι δύσκολη και για αυτό έχει αφιερωθεί αρκετή μελέτη και θα συνεχίσει να αφιερώνεται στο σχεδιασμό αλγορίθμων που βρίσκουν τη βέλτιστη λύση γρήγορα και αξιόπιστα[1].

2.2 Υπερπαράμετροι (Hyperparameters)

Υπερπαράμετροι είναι ένας όρος που πολύ συχνά ακούμε στη μηχανική μάθηση και ο ρόλος τους είναι κρίσιμος για την αύξηση της απόδοσης κατά την εκπαίδευση ενός μοντέλου μηχανικής μάθησης. Τι είναι και γιατί μας απασχολεί;

Ένα μοντέλο μηχανικής μάθησης ορίζεται ως ένα μαθηματικό μοντέλο που περιλαμβάνει παραμέτρους που μαθαίνονται και προσαρμόζονται ανάλογα με τα δεδομένα. Εκπαιδεύοντας ένα μοντέλο με ήδη υπάρχοντα δεδομένα, είμαστε σε θέση να προσαρμόσουμε τις παραμέτρους του μοντέλου. Στους αλγορίθμους μηχανικής μάθησης αυτόματα μαθαίνουν και προσαρμόζουν τις εσωτερικές παραμέτρους βασιζόμενοι στα δεδομένα. Αυτοί οι τύποι παραμέτρων ονομάζονται παράμετροι μοντέλου ή απλώς παράμετροι. Βέβαια υπάρχουν και άλλοι παράμετροι που δεν προσαρμόζονται κατά τη διαδικασία εκμάθησης. Επιλέγονται πριν αρχίσει η εκπαίδευση του μοντέλου. Αυτού του τύπου οι παράμετροι συχνά αναφέρονται ως ”υπερπαράμετροι”. Οι παράμετροι ενός μοντέλου καθορίζουν το πώς τα δεδομένα εισόδου μετατρέπονται στην επιθυμητή έξοδο, ενώ οι υπερπαράμετροι δείχνουν πώς είναι δομημένο το μοντέλο. Ο όρος παράμετροι αναφέρεται στον προγραμματισμό ως ορίσματα μιας συνάρτησης, ενώ οι υπερπαράμετροι καθορίζουν πως οι αλγόριθμοι δουλεύουν και έχουν άμεση επίδραση στο τελικό αποτέλεσμα[3]. Η απόδοση του μοντέλου μηχανικής μάθησης μπορεί να αλλάξει δραστικά ανάλογα με την επιλογή των τιμών των υπερπαραμέτρων του[4].

Για παράδειγμα, στον αλγόριθμο ”δέντρο αποφάσεων” (decision tree), μια από τις υπερπαραμέτρους του είναι το βάθος δέντρου (tree_depth). Ένα δέντρο απόφασης είναι μια δομή δέντρου τύπου διαγράμματος ροής, όπου κάθε εσωτερικός κόμβος υποδηλώνει μια δοκιμή σε ένα χαρακτηριστικό, κάθε κλάδος αντιπροσωπεύει ένα αποτέλεσμα της δοκιμής και κάθε κόμβος φύλλου (τερματικός κόμβος) έχει μια ετικέτα κλάσης. Η παράμετρος `"tree_depth"` ορίζει το όριο για να σταματήσει η περαιτέρω διαίρεση των κόμβων όταν έχει επιτευχθεί το καθορισμένο βάθος

δέντρου κατά τη διάρκεια της κατασκευής του αρχικού δέντρου απόφασης. Η επιλογή μιας μέτριας τιμής για αυτήν την υπερπαράμετρο μπορεί να επιτύχει καλά αποτελέσματα, ενώ μια υψηλή τιμή μπορεί να μειώσει την απόδοση του αλγορίθμου. Για το λόγο αυτό, θα πρέπει να οριστούν οι υπερπαράμετροι πολύ προσεκτικά[4]. Η επιλογή των προκαθορισμένων τιμών δεν εγγυάται πάντα την καλύτερη απόδοση[3]. Για να αυξήσουμε λοιπόν την απόδοση ενός μοντέλου θα πρέπει οι υπερπαράμετροι του να βελτιστοποιηθούν. Με λίγα λόγια καλούμαστε να λύσουμε ένα πρόβλημα βελτιστοποίησης.

2.3 Πρόβλημα εύρεσης βέλτιστων υπερπαραμέτρων

Κατανοώντας το πρόβλημα βελτιστοποίησης και τι είναι οι υπερπαράμετροι μπορούμε εύκολα να καταλάβουμε ότι το πρόβλημα βελτιστοποίησης υπερπαραμέτρων δεν είναι τίποτα άλλο από ένα πρόβλημα βελτιστοποίησης. Η διαδικασία της επιλογής των βέλτιστων υπερπαραμέτρων ονομάζεται βελτιστοποίηση υπερπαραμέτρων. Η ερώτηση στο ποια είναι η πιο αποδοτική προσέγγιση στη βελτιστοποίηση υπερπαραμέτρων έχει συζητηθεί πολλές φορές. Ο σκοπός είναι να χρησιμοποιηθούν όσο το δυνατόν λιγότεροι υπολογιστικοί πόροι για τη βέλτιστη εύρεση τους [3].

Ένα πρόβλημα βελτιστοποίησης υπερπαραμέτρων επίσης μπορεί να θεωρηθεί ως ένα πρόβλημα βελτιστοποίησης μαύρου κουτιού. Πράγματι, το πρόβλημα βελτιστοποίησης υπερπαραμέτρων είναι ισοδύναμο με ένα μαύρο κουτί που παίρνει τις υπερπαραμέτρους ενός δεδομένου αλγορίθμου και επιστρέφει κάποιο μέτρο απόδοσης, καθορισμένο εκ των προτέρων, όπως ο χρόνος μέχρι τη λύση, η τιμή του καλύτερου σημείου που βρέθηκε ή ο αριθμός των λυμένων προβλημάτων. Για παράδειγμα, στην περίπτωση των νευρωνικών δικτύων, το μαύρο κουτί μπορεί να επιστρέψει την ακρίβεια στο δοκιμαστικό σύνολο δεδομένων ως μέτρο απόδοσης[5].

Λαμβάνοντας υπόψη τους υπολογιστικούς πόρους που απαιτούνται, υπερπαράμετροι με μεγαλύτερη σημασία λαμβάνουν προνομιακή μεταχείριση στη διαδικασία εύρεσης βέλτιστων υπερπαραμέτρων. Πολλές φορές είναι δύσκολο να προσδιοριστεί ποσοτικά ποιες από τις υπερπαραμέτρους είναι οι πιο σημαντικές.

Οι υπερπαράμετροι είναι σημαντικοί για τους αλγόριθμους μηχανικής μάθησης, καθώς ελέγχουν άμεσα τη συμπεριφορά των αλγορίθμων εκπαίδευσης και έχουν σημαντική επίδραση στην απόδοση των μοντέλων μηχανικής μάθησης. Πολλές τεχνικές

έχουν αναπτυχθεί και εφαρμοστεί με επιτυχία για ορισμένους τομείς εφαρμογών. Ωστόσο, αυτή η δουλειά απαιτεί επαγγελματική γνώση και εξειδικευμένη εμπειρία. Και μερικές φορές πρέπει να καταφύγει στην αναζήτηση της ωμής βίας. Επομένως, εάν μπορεί να αναπτυχθεί ένας αποτελεσματικός αλγόριθμος βελτιστοποίησης υπερπαραμέτρων για τη βελτιστοποίηση οποιασδήποτε μεθόδου μηχανικής εκμάθησης, θα βελτιώσει σημαντικά την αποτελεσματικότητα της μηχανικής μάθησης[6].

Στις επόμενες ενότητες θα αναφερθούν μέθοδοι που συναντάμε συχνότερα για την προσέγγιση αυτού του προβλήματος.

2.4 Μη αυτοματοποιημένη αναζήτηση βέλτιστων υπερπαραμέτρων

Στη συγκεκριμένη μέθοδο απλώς επιλέγονται οι τιμές των υπερπαραμέτρων χειροκίνητα χωρίς τη βοήθεια κάποιας αυτοματοποιημένης μεθόδου. Αυτό εξαρτάται από τη διαίσθηση και την εμπειρία των ειδικών που μπορούν να προσδιορίσουν τις σημαντικές παραμέτρους που έχουν μεγαλύτερο αντίκτυπο στα αποτελέσματα και στη συνέχεια να καθορίσουν τη σχέση μεταξύ ορισμένων παραμέτρων και των τελικών αποτελεσμάτων μέσω των εργαλείων οπτικοποίησης. Η μη αυτόματη αναζήτηση απαιτεί από τους χρήστες να έχουν περισσότερες επαγγελματικές γνώσεις και πρακτική εμπειρία. Επομένως, είναι δύσκολο να εφαρμοστεί από μη ειδικούς χρήστες. Η διαδικασία βελτιστοποίησης υπερπαραμέτρων δεν είναι εύκολα αναπαραγόμενη. Επιπλέον, καθώς αυξάνεται ο αριθμός των υπερπαραμέτρων και το εύρος των τιμών, γίνεται αρκετά δύσκολο να το διαχειριστούμε, καθώς οι άνθρωποι δεν είναι καλοί στο χειρισμό δεδομένων υψηλών διαστάσεων και εύκολα παρερμηνεύουν ή χάνουν τις τάσεις και τις σχέσεις που έχουν οι υπερπαράμετροι [6].

2.5 Αυτοματοποιημένη αναζήτηση βέλτιστων υπερπαραμέτρων

Για να ξεπεραστούν τα μειονεκτήματα της μη αυτόματης αναζήτησης, αλγόριθμοι αυτοματοποιημένης αναζήτησης έχουν προταθεί. Δηλαδή αλγόριθμοι που παράγουν τις τιμές των υπερπαραμέτρων και πραγματοποιούν δοκιμές αυτόματα για το εκάστοτε πρόβλημα. Υπάρχουν διάφοροι μέθοδοι, όπως για παράδειγμα η τυχαία αναζήτηση, που είναι αρκετά χρήσιμοι σε πολυδιάστατους χώρους αναζήτησης

υπερπαραμέτρων. Οι αλγόριθμοι προσαρμοστικής αναζήτησης προσφέρουν υπολογιστικά αποδοτικές λύσεις σε δύσκολα προβλήματα βελτιστοποίησης στα οποία είναι διαθέσιμη ελάχιστη γνώση σχετικά με τους βέλτιστους υποχώρους. Προσεγγίσεις όπως η Μπεϋζιανή βελτιστοποίηση χρησιμοποιούνται ευρέως για τη βελτιστοποίηση μοντέλων μαύρου κουτιού[3]. Παρακάτω θα δούμε κάποιους από τους πιο κοινούς μεθόδους για τη λύση αυτού του προβλήματος.

2.5.1 Αλγόριθμοι δειγματοληψίας (Sampling algorithms)

Ένας αλγόριθμος δειγματοληψίας είναι μια διαδικασία που μας επιτρέπει να επιλέξουμε τυχαία ένα υποσύνολο από έναν πληθυσμό χωρίς να απαριθμήσουμε όλα τα πιθανά δείγματα του πληθυσμού. Σε πολλά προβλήματα δειγματοληψίας, ο αριθμός των πιθανών δειγμάτων είναι γενικά πολύ μεγάλος. Η επιλογή ενός δείγματος με την απαρίθμηση όλων των πιθανών δειγμάτων είναι γενικά αδύνατη. Ένας αλγόριθμος δειγματοληψίας είναι μια μέθοδος που επιτρέπει την παράκαμψη της απαρίθμησης όλων των πιθανών δειγμάτων [7].

Σε αυτές τις μεθόδους πραγματοποιούνται δοκιμές χωρίς να έχουμε κάποια εξάρτηση μεταξύ τους. Επειδή κάθε δοκιμή γίνεται απομονωμένα είναι εύκολο οι μέθοδοι να παραλληλοποιηθούν. Αλλά δεν έχουμε τη δυνατότητα να χρησιμοποιήσουμε τα αποτελέσματα από τις προηγούμενες δοκιμές για την εξαγωγή συμπερασμάτων για την επόμενη δοκιμή. Παρακάτω θα δούμε τους πιο κοινούς μεθόδους σε αυτή την κατηγορία.

Αλγόριθμος αναζήτησης πλέγματος (Grid search)

Ο αλγόριθμος αναζήτησης πλέγματος είναι μια μέθοδος εξαντλητικής αναζήτησης. Η αναζήτηση πλέγματος παράγει κάθε συνδυασμό πιθανών τιμών υπερπαραμέτρων στο σύνολο της εκπαίδευσης. Στο τέλος, η αναζήτηση πλέγματος εξάγει υπερπαραμέτρους της καλύτερης λύσης που βρέθηκε. Αν και αυτή η μέθοδος επιτυγχάνει αυτοματοποιημένη εύρεση βέλτιστων υπερπαραμέτρων και μπορεί θεωρητικά να βρει τη βέλτιστη τιμή της συνάρτησης στόχου βελτιστοποίησης, πάσχει από την κατάρα της διάστασης, δηλαδή η απόδοση του αλγορίθμου μειώνεται γρήγορα καθώς ο αριθμός των υπερπαραμέτρων που βελτιστοποιούνται και το εύρος τιμών τους αυξάνονται[6].

Αλγόριθμος τυχαίας αναζήτησης

Για να λύσουμε το πρόβλημα κόστους του εξαντλητικού αλγορίθμου αναζήτησης πλέγματος, έχει προταθεί ο αλγόριθμος τυχαίας αναζήτησης. Η συνολική απόδοση μπορεί να βελτιωθεί μειώνοντας την αναζήτηση σε υπερπαραμέτρους που δεν έχουν σημασία. Η τυχαία αναζήτηση δοκιμάζει τυχαίους συνδυασμούς ενός εύρους τιμών. Σε σύγκριση με την αναζήτηση πλέγματος, η τυχαία αναζήτηση είναι πιο αποτελεσματική σε χώρους υψηλών διαστάσεων. Ωστόσο, φαίνεται ότι η τυχαία αναζήτηση είναι αναξιόπιστη για την εκπαίδευση ορισμένων πολύπλοκων μοντέλων[6].

2.5.2 Μπεϋζιανή βελτιστοποίηση (Bayesian optimization)

Μια πολύ γνώστη μέθοδος που χρησιμοποιείται για την εύρεση βέλτιστων υπερπαραμέτρων είναι η Μπεϋζιανή βελτιστοποίηση γνωστή και ως διαδοχική βελτιστοποίηση βάσει μοντέλων. Η διαφορά της Μπεϋζιανής βελτιστοποίησης από άλλες μεθόδους είναι ότι κατασκευάζει ένα πιθανοτικό μοντέλο για τη συνάρτηση που είναι προς βελτιστοποίηση και στη συνέχεια εκμεταλλεύεται αυτό το μοντέλο για να λάβει αποφάσεις σχετικά με το ποιες θα είναι οι επόμενες τιμές υπερπαραμέτρων για την επόμενη αξιολόγηση, ενώ ενσωματώνει την αβεβαιότητα. Η βασική φιλοσοφία είναι η χρήση όλων των πληροφοριών που διατίθεται από προηγούμενες αξιολογήσεις της συνάρτησης. Αυτό οδηγεί σε μια διαδικασία που μπορεί να βρει βέλτιστες υπερπαραμέτρους με σχετικά λίγες αξιολογήσεις, με κόστος όμως τη χρησιμοποίηση περισσότερων υπολογιστικών πόρων για την επιλογή των υπερπαραμέτρων της επόμενης δοκιμής. Όταν οι αξιολογήσεις της συνάρτησης είναι δαπανηρές στην εκτέλεση τους — όπως συμβαίνει κατά την εκπαίδευση ενός αλγόριθμου μηχανικής μάθησης — τότε είναι εύκολο να δικαιολογηθούν κάποιοι επιπλέον υπολογισμοί για τη λήψη καλύτερων αποφάσεων[8].

Υπάρχουν δύο σημαντικές επιλογές που πρέπει να γίνουν κατά την εκτέλεση της Μπεϋζιανής βελτιστοποίησης. Πρώτα θα πρέπει να επιλεχθεί η μέθοδος που θα δημιουργεί μοντέλα που θα εκφράζουν υποθέσεις για τη συνάρτηση που βελτιστοποιείται. Μια μέθοδος που συνήθως επιλέγεται είναι η Γκαουσιανή διαδικασία (Gaussian process). Η Γκαουσιανή διαδικασία είναι μια στοχαστική διαδικασία που είναι γενικά μια συλλογή τυχαίων μεταβλητές με δείκτες βασισμένοι κατά χρόνο ή χώρο. Η ιδιαίτερη ιδιότητά του είναι ότι κάθε πεπερασμένη συλλογή από αυτές τις

μεταβλητές είναι ότι ακολουθούν μια πολυμεταβλητή κατανομή Gauss. Έτσι, η Γκαουσιανή διαδικασία είναι μια κατανομή πάνω σε άπειρες μεταβλητές και, επομένως, μια κατανομή σε συναρτήσεις με συνεχής πεδίο ορισμού. Κατά συνέπεια, περιγράφει μια κατανομή πιθανότητας σε μια άπειρη διάσταση διανυσματικού χώρου[9].

Επίσης, πρέπει να επιλεγεί μια συνάρτηση εκμάθησης (acquisition function). Οι συναρτήσεις εκμάθησης είναι μαθηματικές τεχνικές που καθοδηγούν τον τρόπο με τον οποίο πρέπει να διερευνηθεί ο χώρος των παραμέτρων κατά τη διάρκεια της Μπεϋζιανής βελτιστοποίησης. Χρησιμοποιούν τον προβλεπόμενο μέσο όρο και την προβλεπόμενη διακύμανση που δημιουργούνται από το μοντέλο διαδικασίας Γκάους. Για ένα σύνολο τέτοιων προβλέψεων σε ένα σύνολο υποψήφιων συνόλων παραμέτρων, μια συνάρτηση εκμάθησης συνδυάζει τα μέσα και τις αποκλίσεις σε ένα κριτήριο που θα κατευθύνει την αναζήτηση.

Ο όρος διακύμανσης που δημιουργείται από το μοντέλο διαδικασίας Γκάους συνήθως αντανακλά τις χωρικές πτυχές των δεδομένων. Τα υποψήφια σύνολα με υψηλή διακύμανση δεν βρίσκονται κοντά σε καμία υπάρχουσα τιμή παραμέτρων (δηλαδή εκείνα που έχουν παρατηρήσει εκτιμήσεις απόδοσης). Η προβλεπόμενη διακύμανση είναι πολύ κοντά στο μηδέν ή πολύ κοντά σε ένα υπάρχον [10].

Δύο στρατηγικές που υπάρχουν για τη συνάρτηση εκμάθησης είναι οι εξής:

- **Αξιοποίηση**: εστιάζει σε αποτελέσματα κοντά στα τρέχοντα καλύτερα αποτελέσματα, τιμωρώντας για υψηλότερες τιμές διακύμανσης.
- **Εξερεύνηση**: ωθεί την αναζήτηση προς ανεξερεύνητες περιοχές του χώρου των παραμέτρων.

2.5.3 Εξελικτική υπολογιστική

Μια ακόμα κατηγορία που έχει προταθεί είναι η εξελικτική υπολογιστική. Η εξελικτική υπολογιστική περιέχει διάφορους αλγορίθμους που ονομάζονται εξελικτικοί αλγόριθμοι. Τις τελευταίες δύο δεκαετίες οι εξελικτικοί αλγόριθμοι έχουν γίνει πολύ δημοφιλές εργαλείο για αναζήτηση, βελτιστοποίηση και παροχή λύσεων σε σύνθετα προβλήματα.

Οι εξελικτικοί αλγόριθμοι βασίζονται στην αρχή της επιβίωσης του καταλληλότερου. Είναι ιδιαίτερα εμπνευσμένοι από τη Δαρβινική εξελικτική έννοια που αλλάζει το σύστημα σταδιακά με την πάροδο του χρόνου. Στη φύση, τα άτομα πρέπει να

προσαρμοστούν στο περιβάλλον τους για να επιβιώσουν, αυτή η διαδικασία είναι γνωστή ως εξέλιξη. Ο χρόνος αναπαραγωγής, τα χαρακτηριστικά κάθε ατόμου που το καθιστούν πιο κατάλληλο να ανταγωνίζεται, διατηρείται και τα ασθενέστερα στοιχεία εξαλείφονται. Τα γονίδια είναι οι μονάδες που ελέγχουν αυτά τα χαρακτηριστικά, ένα σύνολο τέτοιων γονιδίων σχηματίζει χρωμοσώματα. Μόνο οι πιο δυνατοί επιβιώνουν στις επόμενες γενιές και τα πιο κατάλληλα γονίδια μεταδίδονται στους απογόνους τους κατά τη διάρκεια της διαδικασίας γενετικού ανασυνδυασμού. Αυτή η διαδικασία φυσικής επιλογής και βελτιστοποίησης οδηγεί στην ανάπτυξη των «εξελικτικών αλγορίθμων».

Ένας εξελικτικός αλγόριθμος είναι εμπνευσμένος από το μηχανισμό της βιολογικής εξέλιξης, όπως η αναπαραγωγή, η μετάλλαξη, ο ανασυνδυασμός και η επιλογή. Ένα σύνολο υποψήφιων λύσεων δημιουργείται τυχαία για να μεγιστοποιηθεί η συνάρτηση ποιότητας. Τότε η συνάρτηση ποιότητας με τη μορφή αφηρημένης συνάρτησης καταλληλότητας εφαρμόζεται στον τομέα του προβλήματος. Στην επόμενη γενιά επιλέγονται καλύτεροι υποψήφιοι με βάση τη συνάρτηση καταλληλότητας. Αυτό επιτυγχάνεται με την εφαρμογή της τεχνικής του ανασυνδυασμού ή/και μετάλλαξη σε αυτά. Ο ανασυνδυασμός αντιπροσωπεύεται από το δυαδικό τελεστή. Αυτός ο τελεστής μπορεί να εφαρμοστεί σε δύο ή περισσότερους επιλεγμένους υποψήφιους γνωστοί ως γονείς που στη συνέχεια δημιουργούν ένα ή περισσότερους νέους υποψήφιους (παιδιά) ως αποτέλεσμα. Ενώ εφαρμόζεται μετάλλαξη σε έναν μόνο υποψήφιο και καταλήγει σε ένα νέο παιδί. Μετά εκτελώντας τον ανασυνδυασμό ή μετάλλαξη, δημιουργεί ένα σύνολο από νέους υποψηφίους με βάση τη τη συνάρτηση καταλληλότητας. Αυτή είναι μια επαναληπτική διαδικασία και μπορεί να συνεχιστεί μέχρι να η επιθυμητή λύση[11]. Στο Σχήμα 2.1 μπορούμε να δούμε συνοπτικά τη διαδικασία που περιγράφηκε.

Παρακάτω θα δούμε δύο εξελικτικούς αλγορίθμους που έχουν χρησιμοποιηθεί για την επίλυση του προβλήματος εύρεσης βέλτιστων υπερπαραμέτρων.

Γενετικός αλγόριθμος

Ο γενετικός αλγόριθμος (GA) βασίζεται στην έννοια της φυσικής επιλογής. Ο GA διατηρεί έναν πληθυσμό πιθανών λύσεων στο πρόβλημα βελτιστοποίησης, οι οποίες εξελίσσονται σε πολλές γενιές προκειμένου να παραχθεί η καλύτερη λύση.

Σχήμα 2.1: Μια γενική εικόνα τρόπου λειτουργίας για έναν απλό εξελικτικό αλγόριθμο

Κάθε πιθανή λύση αναφέρεται ως χρωμόσωμα. Κάθε χρωμόσωμα αντιπροσωπεύει ένα σημείο στο χώρο των υπερπαραμέτρων. Η ύπαρξη πολλαπλών χρωμοσωμάτων επιτρέπει στο GA να εξερευνήσει πολλαπλές λύσεις παράλληλα. Ο αριθμός των γονιδίων σε ένα χρωμόσωμα ταιριάζει με τη διάσταση του χώρου υπερπαραμέτρων. Η εξέλιξη προς την καλύτερη λύση είναι επαναληπτική. Κάθε επανάληψη αντιστοιχεί σε μία γενιά στην εξέλιξη όλων των χρωμοσωμάτων και αποτελείται από 3 διακριτά στάδια: την επιλογή των γονέων, τη διασταύρωση των γονιδίων και τη μετάλλαξη.

Η επιλογή των γονέων γίνεται με τη μέθοδο του τουρνουά. Σε κάθε τουρνουά ένας συγκεκριμένος αριθμός χρωμοσωμάτων ανταγωνίζεται για να επιλεγεί ως γο-

νέας για την επόμενη γενιά. Ο αριθμός των χρωμοσωμάτων που συμμετέχουν σε κάθε τουρνουά συμβολίζεται με το σύμβολο:

$$Ntour$$

. Οι συμμετέχοντες προέρχονται από τον πληθυσμό των χρωμοσωμάτων τυχαία και κατατάσσονται κατά σειρά φθίνουσας με βάση τη βαθμολογία. Ο συμμετέχων με την υψηλότερη βαθμολογία επιλέγεται ως γονέας με πιθανότητα:

$$Ptour$$

. Σε περίπτωση που δεν επιλεγεί το χρωμόσωμα με την υψηλότερη βαθμολογία, επιλέγεται το χρωμόσωμα με τη δεύτερη υψηλότερη βαθμολογία, και πάλι με την πιθανότητα $Ptour$, και ούτω καθεξής. Το τουρνουά τελειώνει όταν επιλεχθούν με αυτόν τον τρόπο δύο χρωμοσώματα για να είναι οι γονείς.

Μια μεγαλύτερη τιμή του $Ntour$ έχει ως αποτέλεσμα το χρωμόσωμα με χαμηλή βαθμολογία να έχει μικρότερες πιθανότητες να επιλεγεί ως γονέας για την επόμενη γενιά, επειδή υπάρχει μεγάλη πιθανότητα ένα χρωμόσωμα με καλύτερη βαθμολογία να συμμετέχει στο ίδιο τουρνουά. Για μικρότερη τιμή του $Ntour$ έχει το αντίθετο αποτέλεσμα. Ξεκινούν νέα τουρνουά μέχρι να επιλεγεί επαρκής αριθμός ζευγαριών γονέων για την παραγωγή των χρωμοσωμάτων για την επόμενη γενιά.

Τα χρωμοσώματα από δύο γονείς παράγουν ένα νέο χρωμόσωμα για την επόμενη γενιά μέσω ανασυνδυασμού. Χρησιμοποιείται ανασυνδυασμός κ-σημείων στην οποία τα χρωμοσώματα και των δύο γονέων κόβονται σε k σημεία ($Ncross$ αναφέρεται στον αριθμό των σημείων) και τα χρωμοσώματα των απογόνων παράγονται με τυχαία επιλογή τμημάτων χρωμοσωμάτων από οποιονδήποτε γονέα. Τα χρωμοσώματα των απογόνων που λαμβάνονται με τη λειτουργία διασταύρωσης υπόκεινται σε μετάλλαξη, η οποία στοχεύει στην αύξηση της ποικιλομορφίας του πληθυσμού, επιτρέποντας έτσι την εξερεύνηση περιοχών στο χώρο των υπερπαραμέτρων, που δεν περιέχουν χρωμοσώματα από τη μητρική γενιά. Η μετάλλαξη βοηθά επίσης να αποφευχθεί ο πληθυσμός να κολλήσει στα τοπικά ελάχιστα[12].

Βελτιστοποίηση σμήνους σωματιδίων (Particle Swarm Optimization

Η βελτιστοποίηση σμήνους σωματιδίων (Particle Swarm Optimization - PSO) αντιπροσωπεύει μια υπολογιστική μέθοδο για τη βελτιστοποίηση συνεχών μη γραμμικών συναρτήσεων. Όπως υποδηλώνει το όνομα της μεθόδου, η μεγιστοποίηση της αντικειμενικής συνάρτησης από το PSO εκτελείται από ένα σμήνος σωματιδίων. Τα σωματίδια διασχίζουν το χώρο των υπερπαραμέτρων, με τη θέση κάθε σωματιδίου να αντιπροσωπεύει ένα σύνολο υπερπαραμέτρων. Η ύπαρξη ενός σμήνος σωματιδίων επιτρέπει την παράλληλη εξερεύνηση πολλαπλών σημείων στο χώρο των υπερπαραμέτρων, επιτρέποντας έτσι μια εξαιρετικά παράλληλη υλοποίηση του αλγορίθμου PSO σε έναν υπολογιστή. Η εξέλιξη του σμήνους σωματιδίων προχωρά σε επαναλήψεις που υποδηλώνονται με το γράμμα k. Σε κάθε επανάληψη μια νέα θέση x_i^{k+1} υπολογίζεται για κάθε σωματίδιο i σύμφωνα με τη σχέση:

$$x_i^{k+1} = x_i^k + wp_i^k + F_i^k$$

όπου x_i^k δηλώνει την τρέχουσα θέση του σωματιδίου και p_i^k την ορμή του. Ο όρος ορμής wp_i^k αντιπροσωπεύει την αδράνεια για τα σωματίδια να αλλάζουν την κατεύθυνση τους όταν διασχίζουν το χώρο των υπερπαραμέτρων. Το σύμβολο F_i^k αντιπροσωπεύει μια ελκτική δύναμη, η οποία έχει ως αποτέλεσμα τα σωματίδια να κινούνται προς τα άκρα της αντικειμενικής συνάρτησης που είχαν ανακαλυφθεί προηγουμένως. Ο όρος ορμής προκαλεί μια τάση για τα σωματίδια να συνεχίσουν να κινούνται προς την τρέχουσα κατεύθυνσή τους, περνώντας από τα προηγούμενα άκρα. Αυτή η συμπεριφορά αυξάνει την εξερεύνηση του χώρου υπερπαραμέτρων Η και βρέθηκε ότι βελτιώνει την απόδοση. Ο συντελεστής w αναφέρεται ως αδρανειακό βάρος στη βιβλιογραφία[12].

2.5.4 Βελτιστοποίηση χωρίς παράγωγο

Είναι το πεδίο που στοχεύει στην επίλυση προβλημάτων βελτιστοποίησης όπου οι παράγωγοι δεν είναι διαθέσιμοι, αν και μπορεί να υπάρχουν. Για παράδειγμα, όταν οι συναρτήσεις στόχου και/ή περιορισμού είναι μη διαφοροποιήσιμες, θορυβώδεις ή δαπανηρές στην αξιολόγησή τους. Επιπλέον, η αξιολόγηση σε ορισμένα σημεία μπορεί να αποτύχει. Βελτιστοποίηση μαύρου κουτιού είναι μια υποκατη-

γορία της βελτιστοποίησης χωρίς παράγωγο όπου οι παράγωγοι δεν υπάρχουν και το πρόβλημα μοντελοποιείται ως ένα μαύρο κουτί. Αυτός ο όρος αναφέρεται στο γεγονός ότι η υπολογιστική διαδικασία πίσω από τις τιμές εξόδου είναι άγνωστη.Το γενικό πρόβλημα βελτιστοποίησης χωρίς παράγωγο περιγράφεται ως

$$minf(x), x \in \Omega$$

όπου f είναι η συνάρτηση που ελαχιστοποιείται στο χώρο Ω[5].

Υπάρχουν δύο κύριες κατηγορίες μεθόδων βελτιστοποίησης χωρίς παράγωγο: μέθοδοι αναζήτησης βάσει μοντέλου και άμεσης αναζήτησης. Η πρώτη μέθοδος χρησιμοποιεί την τιμή του στόχου ή/και τους περιορισμούς σε ορισμένα ήδη αξιολογημένα σημεία για να δημιουργήσει ένα μοντέλο ικανό να καθοδηγήσει τη βελτιστοποίηση βασιζόμενος στις προβλέψεις του μοντέλου. Για παράδειγμα, περιλαμβάνει μεθόδους που βασίζονται σε περιοχές εμπιστοσύνης ή μοντέλα παρεμβολής. Αυτό τις διαφοροποιεί από τις μεθόδους άμεσης αναζήτησης που υιοθετούν μια πιο απλή στρατηγική βελτιστοποίηση του μαύρου κουτιού. Σε κάθε επανάληψη, οι μέθοδοι άμεσης αναζήτησης δημιουργούν ένα σύνολο δοκιμαστικών σημείων που συγκρίνονται με την καλύτερη διαθέσιμη λύση. Για παράδειγμα, ο αλγόριθμος Generalized Pattern Search (GPS) ορίζει ένα πλέγμα στο χώρο αναζήτησης και καθορίζει το επόμενο σημείο προς αξιολόγηση επιλέγοντας μια κατεύθυνση αναζήτησης. Οι αλγόριθμοι βελτιστοποίησης χωρίς παράγωγο συνήθως περιλαμβάνουν μια απόδειξη σύγκλισης που εξασφαλίζει μια λύση καλής ποιότητας κάτω από ορισμένες υποθέσεις για την αντικειμενική συνάρτηση[5].

Κεφάλαιο 3

Υλοποίηση

Σε αυτό το κεφάλαιο θα γίνει η περιγραφή των αλγόριθμων που υλοποιήθηκαν και των προγραμμάτων που χρησιμοποιήθηκαν στο θεωρητικό τους κομμάτι αλλά και στο πρακτικό τους. Για το πρακτικό κομμάτι θα χρησιμοποιηθεί και ένα απλό παράδειγμα βελτιστοποίησης παραμέτρων. Οι αλγόριθμοι υλοποιήθηκαν με τη γλώσσα προγραμματισμού Python. Τα αποτελέσματα και οι δοκιμές που θα γίνουν είναι βασισμένες στους κώδικες που υλοποιήθηκαν στα πλαίσια της διπλωματικής.

3.1 Απλό παράδειγμα βελτιστοποίησης

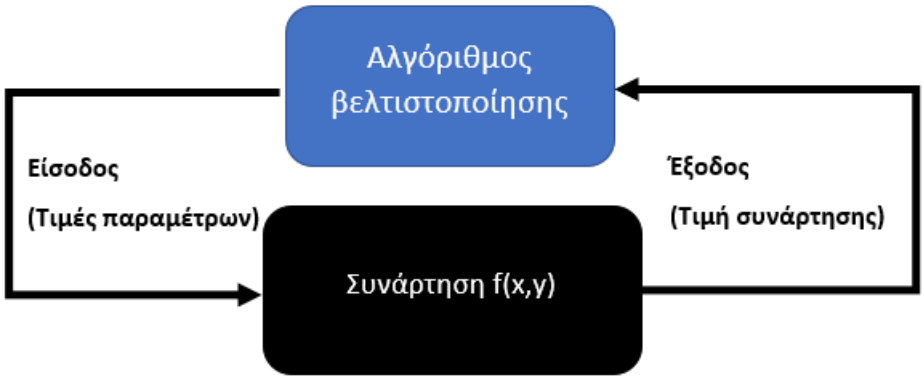
Έστω έχουμε τη συνάρτηση

$$f(x,y) = \frac{x^2+y^2}{y} x \in (0,1), y \in (1,2)$$

$$x, y \in \mathbb{R}$$

. Ο σκοπός μας είναι να βρούμε το κατάλληλο συνδυασμό των x, y ώστε η συνάρτηση f να επιστρέψει τη μικρότερη τιμή, δηλαδή μιλάμε για ένα πρόβλημα βελτιστοποίησης εύρεσης παραμέτρων όπου προσπαθούμε να ελαχιστοποιήσουμε τη συνάρτησή μας. Όπως περιγράφηκε προηγουμένως ένα πρόβλημα βελτιστοποίησης μπορούμε να το δούμε και ως ένα πρόβλημα μαύρου κουτιού. Δηλαδή δε ξέρουμε ποια είναι η συνάρτηση μας, το μόνο που ξέρουμε είναι οι παράμετροι και οι περιορισμοί τους. Επίσης, επειδή οι παράμετροι είναι πραγματικοί και μπορούν να πάρουν οποιεσδήποτε τιμές εντός των διαστημάτων, άρα υπάρχουν άπειροι συνδυασμοί, εμείς για το παράδειγμα μας θα κάνουμε 10 δοκιμές για κάθε αλγόριθμο και πρόγραμμα.

Όπως βλέπουμε στο Σχήμα 3.1 έχουμε το πρόβλημα ως ένα μαύρο κουτί (συνάρ-



Σχήμα 3.1: Περιγραφή προβλήματος ως ένα μαύρο κουτί

τηση f) και έναν αλγόριθμο βελτιστοποίησης που αναλαμβάνει να βρει τη μικρότερη τιμή της συνάρτησης κάνοντας δοκιμές με συνδυασμούς των παραμέτρων της. Παρακάτω θα δούμε τους αλγορίθμους που χρησιμοποιήθηκαν στη διπλωματική λύνοντας αυτό το απλό πρόβλημα.

3.2 Αλγόριθμος αναζήτησης πλέγματος (Grid search)

Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο ο αλγόριθμός αναζήτησης πλέγματος είναι ένας εξαντλητικός αλγόριθμος, δηλαδή δοκιμάζει όλους τους πιθανούς συνδυασμούς του χώρου αναζήτησης των παραμέτρων. Βέβαια αυτό δεν είναι πάντα δυνατό καθώς όταν έχουμε πραγματικές παραμέτρους τότε υπάρχουν άπειροι συνδυασμοί. Η δοκιμή άπειρων συνδυασμών πρακτικά είναι αδύνατον, για αυτό και ο χρήστης πρέπει να επιλέγει τον αριθμό δοκιμών.

Υλοποίηση αλγορίθμου

Ο βασικός τρόπος λειτουργίας του αλγόριθμου grid search στη συγκεκριμένη υλοποίηση είναι ο εξής: Το πλέγμα δημιουργείται χωρίζοντας τα διαστήματα των υπερπαραμέτρων σε τόσες τιμές, ώστε οι συνδυασμοί να ισούνται με τις δοκιμές που θέλουμε να κάνουμε. Στη συγκεκριμένη υλοποίησή το πλέγμα που δημιουργείται έχει πάντα τις ίδιες διαστάσεις σε κάθε παράμετρο (πχ. 1x1, 2x2, 3x3). Αρκετές φορές όμως επειδή το πλέγμα έχεις τις ίδιες διαστάσεις αυτό μπορεί να μη συμβαδίζει με τον αριθμό δοκιμών που θέλουμε να εκτελέσουμε. Για παράδειγμα αν

έχουμε δυο μεταβλητές x, y και θέλουμε να κάνουμε 9 δοκιμές τότε θα δημιουργηθεί ένα πλέγμα 3x3 που είναι αποδεκτό, αν όμως αντί για 9 δοκιμές θελήσουμε 10 δοκιμές για το ίδιο πρόβλημα, τότε το πλέγμα 3x3 δε μας καλύπτει γιατί θα γίνουν μόνο 9 συνδυασμοί, ενώ η αμέσως επόμενη επιλογή είναι να δημιουργήσουμε ένα πλέγμα 4x4. Το πλέγμα 4x4 δημιουργεί 16 συνδυασμούς, ενώ εμείς θέλουμε 10. Για να επιλύσουμε αυτό το πρόβλημα δημιουργούμε ένα πλέγμα με παραπάνω διαστάσεις άρα περισσότερους συνδυασμούς και απλώς επιλέγουμε τυχαία συνδυασμούς ακριβώς όσο το πλήθος των δοκιμών που θέλουμε. Δηλαδή αφού θέλουμε 10 συνδυασμούς για x, y, θα δημιουργήσουμε 16 συνδυασμούς(πλέγμα 4x4) και θα επιλέξουμε τυχαία τους 10 από αυτούς. Επίσης έχουμε τη δυνατότητα να επιλέξουμε και τον τύπο των μεταβλητών (ακέραιες ή πραγματικές). Αφού γίνουν οι δοκιμές, επιστρέφεται το καλύτερο αποτέλεσμα που βρήκε και οι παράμετροι που οδήγησαν σε αυτό. Στον Αλγόριθμο 1 βλέπουμε την υλοποίηση του αλγορίθμου βελτιστοποίησης υπερπαραμέτρων με τη χρήση του grid search.

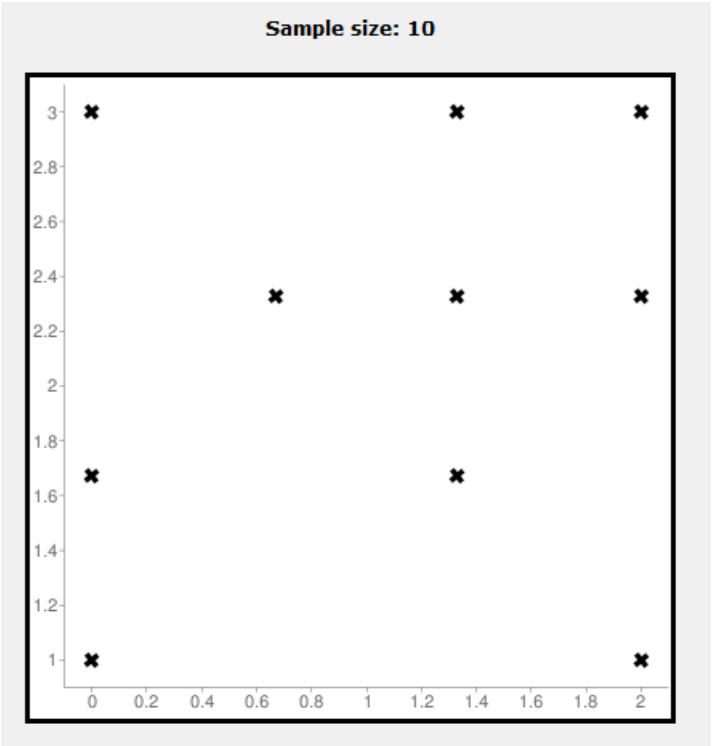
1 Βελτιστοποίηση υπερπαραμέτρων αλγορίθμου αναζήτησης πλέγματος

for $i = 0, i < Number\_of\_Trials, i+ = 1$ **do**
  $delta == i^{Number\_of\_parameters}$
  if $delta = Number\_of\_Trials$ **then**
    for $j = 0, j < Number\_of\_parameters, j+ = 1$ **do**
      Lines[j].add(EventlySpaced(l_bound[j],u_bound[j],Number_of_Trials))
    end for
    combinations=CreateCombinations(Lines)
    break
  end if
  if $delta > Number\_of\_Trials$ **then**
    for $j = 0, j < Number\_of\_parameters, j+ = 1$ **do**
      Lines[j].add(EventlySpaced(l_bound[j],u_bound[j],Number_of_Trials))
    end for
    Temp= CreateCombinations(Lines)
    combinations=Choose_Random(Temp,Number_of_Trials)
    break
  end if
end for
for $i = 0, i < Number\_of\_parameters, i+ = 1$ **do**
  if $type[i] == integer$ **then**
    combinations[i]=round(combinations[i])
  end if
end for
for $i = 0, i < Number\_of\_Trials, i+ = 1)$ **do**
  Current_Result= BlackBox(combinations[i])
  if $CurrentResult < Best\_solution$ **then**
    Best_solution=Current_Result
    Best_parameters=combinations[i]
  end if
end for

- **Number_of_Trials**: Αριθμός δοκιμών.

- **Current_Result**: Αποτέλεσμα τους προβλήματος μετά την εκτέλεσή του.
- **Best_solution**: Η καλύτερη λύση που βρέθηκε.
- **Best_parameters**: Οι παράμετροι που οδήγησαν στην καλύτερη λύση.
- **detla**: Δοκιμαστική μεταβλητή για την εύρεση των διαστάσεων πλέγματος.
- **Lines**: Όλα τα δείγματα κάθε παραμέτρου.
- **l_bound**,**u_bound**: Τα όρια κάθε παραμέτρου.
- **type**: Είδος παραμέτρου (ακέραιος ή πραγματικός).
- **Number_of_parameters**: Το πλήθος των παραμέτρων (διαστάσεις του προβλήματος).
- **EventlySpaced()**: Επιστρέφει έναν αριθμό δειγμάτων σε ομοιόμορφη απόσταση, υπολογισμένα στο διάστημα κάθε παραμέτρου.
- **BlackBox()**: Το πρόβλημα ως προς επίλυση.
- **combinations**: Όλοι οι συνδυασμοί των παραμέτρων.
- **CreateCombinations()**: Δημιουργεί όλους του πιθανούς συνδυασμούς μεταξύ των παραμέτρων.
- **Choose_Random()**: Από όλους τους διαθέσιμους συνδυασμούς επιλέγονται Number_of_Trials συνδυασμοί.

3.2.1 Παράδειγμα εκτέλεσης αλγορίθμου



Σχήμα 3.2: Συνδυασμοί παραμέτρων αλγορίθμου αναζήτησης πλέγματος

Βλέπουμε και στο Σχήμα 3.2 ότι τα δείγματα δεν είναι ομοιόμορφα κατανεμημένα. Αυτό οφείλεται όπως εξηγήθηκε και προηγουμένως στο ότι δημιουργήθηκαν παραπάνω συνδυασμοί (16 στη συγκεκριμένη περίπτωση) και επιλέχθηκαν τυχαία 10 από αυτούς.

Απεικόνιση 3.1: Παράδειγμα αλγορίθμου αναζήτησης πλέγματος

```
{
"f(0.0,3.0) = 3.0 Current best solution 3.0"
"f(2.0,1.0) = 5.0 Current best solution 3.0"
"f(0.67,2.33) = 2.52 Current best solution 2.52"
"f(1.33,3.0) = 3.59 Current best solution 2.52"
"f(1.33,1.67) = 2.73 Current best solution 2.52"
"f(1.33,2.33) = 3.1 Current best solution 2.52"
"f(0.0,1.67) = 1.67 Current best solution 1.67"
"f(2.0,3.0) = 4.33 Current best solution 1.67"
"f(2.0,2.33) = 4.05 Current best solution 1.67"
```

```
"f(0.0,1.0) = 1.0 Current best solution 1.0"

Best parameters: x=0,y=1
Best solution: 1
}
```

Όπως βλέπουμε στην Απεικόνιση 3.1, ο αλγόριθμος πλέγματος αναζήτησης θα πάρει τους συνδυασμούς του πλέγματος που δημιούργησε και θα τους δοκιμάσει. Σε κάθε δοκιμή κρατάμε την τιμή της συνάρτησης και κάνουμε σύγκριση με την καλύτερη λύση που έχει βρεθεί έως τώρα. Στο τέλος αφού ολοκληρωθούν οι δοκιμές μπορούμε να δούμε το αποτέλεσμα καθώς και τις παραμέτρους που μας οδήγησε σε αυτό. Ο τρόπος λειτουργίας του αλγορίθμου grid search μπορεί να μεταφερθεί και σε προβλήματα περισσότερων διαστάσεων. Για παράδειγμα, με 3 παραμέτρους θα είχαμε ένα τρισδιάστατο πλέγμα, για 4 παραμέτρους ένα τετραδιάστατο πλέγμα κτλ.

3.3 Αλγόριθμος τυχαίας αναζήτησης (Random search)

Όπως είχε περιγραφεί ο αλγόριθμος random search στο προηγούμενο κεφάλαιο, ο τρόπος λειτουργίας του είναι πολύ απλώς. Με βάση τα όρια κάθε παραμέτρου παράγει τυχαίες τιμές. Το μεγάλο πλεονέκτημα του είναι ότι μπορούμε να κάνουμε λιγότερες δοκιμές σε σχέση με άλλους αλγορίθμους ελπίζοντας λόγο της τυχαιότητας που εισάγει, ίσως βρούμε τελικά τη βέλτιστη λύση.

Υλοποίηση αλγορίθμου

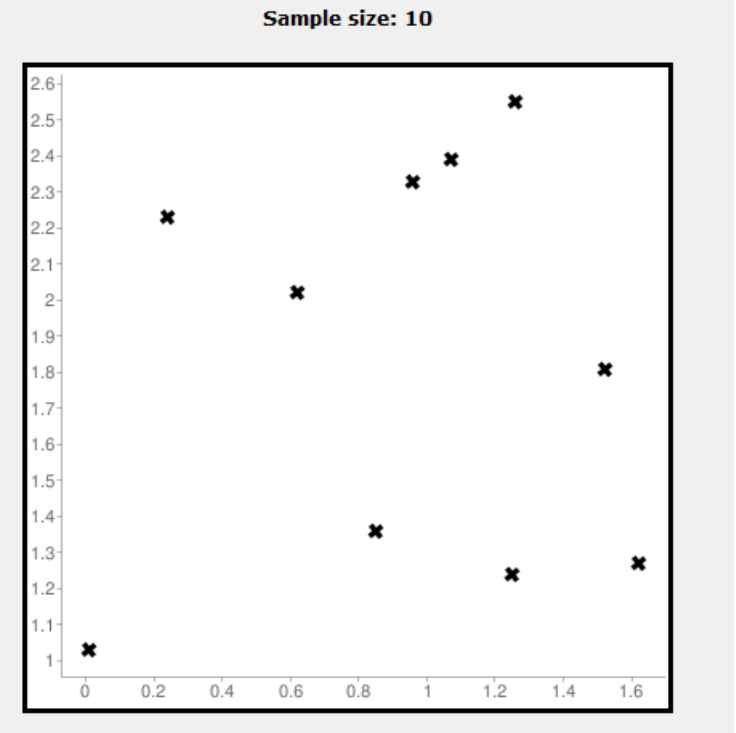
Η υλοποίηση του αλγορίθμου random search είναι πιο απλή. Κάθε φορά δημιουργείται ένας συνδυασμούς τυχαίων τιμών των παραμέτρων και αμέσως μετά γίνεται η δοκιμή. Αυτό συνεχίζει τόσες φορές όσες ο αριθμός δοκιμών που ορίσαμε. Στο τέλος επιστρέφονται οι καλύτερες παράμετροι που οδήγησαν στο καλύτερο αποτέλεσμά. Επίσης όπως και στο grid search έχουμε την επιλογή να ορίσουμε τον τύπο κάθε παραμέτρου. Στον Αλγόριθμο 2 μπορούμε να δούμε τον ενδεικτικό αλγόριθμο για τη συγκεκριμένη υλοποίηση.

2 Βελτιστοποίηση υπερπαραμέτρων αλγορίθμου τυχαίας αναζήτησης

```
for j = 0, j < Number_of_Trials, j = +1 do
  for i = 0, i < len(parameters), i+ = 1 do
    if type[i] == integer then
      combination.add(round(randomgenarator(l_bound[i],u_bound[i])))
    else
      combination.add(randomgenarator(l_bound[i],u_bound[i]))
    end if
    Current_Result= BlackBox(combination)
    if CurrentResult < Best_solution then
      Best_solution=Current_Result
      Best_parameters=combination
    end if
  end for
end for
```

- **Number_of_Trials**: Αριθμός δοκιμών.
- **Current_Result**: Αποτέλεσμα τους προβλήματος μετά την εκτέλεση του.
- **Best_solution**: Η καλύτερη λύση που βρέθηκε.
- **Best_parameters**: Οι παράμετροι που οδήγησαν στην καλύτερη λύση.
- **l_bound**,**u_bound**: Τα όρια κάθε παραμέτρου.
- **type**: Είδος παραμέτρου (ακέραιος ή πραγματικός).
- **Number_of_parameters**: Το πλήθος των παραμέτρων (διαστάσεις του προβλήματος).
- **randomgenarator()**: Δημιουργεί μια τυχαία τιμή με βάση τα όρια της κάθε παραμέτρου.
- **BlackBox()**: Το πρόβλημα ως προς επίλυση.
- **combinations**:: Ο συνδυασμός των παραμέτρων για τη δοκιμή.

3.3.1 Παράδειγμα εκτέλεσης αλγορίθμου



Σχήμα 3.3: Συνδυασμοί παραμέτρων αλγορίθμου τυχαίας αναζήτησης

Ο αλγόριθμος random search όπως αναφέρθηκε και προηγουμένως παράγει τυχαίες τιμές για κάθε παράμετρο. Στο Σχήμα 3.3 βλέπουμε τα τυχαία σημεία που δημιουργήθηκαν από τον αλγόριθμο random search.

Απεικόνιση 3.2: Παράδειγμα αλγορίθμου τυχαίας αναζήτησης

```
{
"f(1.26,2.55) = 3.17 Current best solution 3.17"
"f(0.85,1.36) = 1.89 Current best solution 1.89"
"f(0.01,1.03) = 1.03 Current best solution 1.03"
"f(1.52,1.81) = 3.09 Current best solution 1.03"
"f(1.07,2.39) = 2.87 Current best solution 1.03"
"f(0.62,2.02) = 2.21 Current best solution 1.03"
"f(0.96,2.33) = 2.73 Current best solution 1.03"
"f(0.24,2.23) = 2.26 Current best solution 1.03"
"f(1.25,1.24) = 2.5 Current best solution 1.03"
"f(1.62,1.27) = 3.34 Current best solution 1.03"
```

```
Best Parameters: x=0.01,y=1.03
Best solution: 1.03
}
```

Όπως βλέπουμε στην Απεικόνιση 3.2, ο αλγόριθμος random search θα πάρει τους συνδυασμούς που δημιούργησε και θα τους δοκιμάσει. Σε κάθε δοκιμή κρατάμε την τιμή της συνάρτησης και κάνουμε σύγκριση με την καλύτερη λύση που έχει βρεθεί έως τώρα. Στο τέλος αφού ολοκληρωθούν οι δοκιμές μπορούμε να δούμε το αποτέλεσμα καθώς και τις παραμέτρους που μας οδήγησαν σε αυτό. Ο τρόπος λειτουργίας του αλγορίθμου random search μπορεί να μεταφερθεί και σε προβλήματα περισσότερων διαστάσεων.

3.4 Ακολουθίες χαμηλής απόκλισης

Η απόκλιση σε μια ακολουθία είναι μια μέτρηση που χαρακτηρίζεται ως το πόσο καλή ομοιομορφία υπάρχει σε μια ακολουθία, ή απλώς το πόσο καλά είναι "απλωμένες" οι τιμές της. Επομένως μια ακολουθία χαμηλής απόκλισης είναι μια ακολουθία που τα δείγματα της έχουν χαμηλή απόκλιση μεταξύ τους. Οι καλύτερες ακολουθίες που είναι γνωστές είναι ασυμπτωτικά $O((\log^s n)/n)$ και $O((\log^(s-1)n)/n)$ για πεπερασμένη ακολουθία μεγέθους n. Σύμφωνα με τον Niederreiter[13], η απόκλιση μιας ομοιόμορφης τυχαίας κατανομής είναι περίπου $O(1/\sqrt{n})$ κατά μέσο όρο. Σημειώνουμε ότι ένα n-κανονικό πλέγμα έχει επίσης απόκλιση τάξης $O(1/\sqrt{n})$, που δεν είναι και χαμηλό. Γραφικά, ακολουθία χαμηλής απόκλισης μπορεί να χαρακτηριστεί από το γεγονός ότι οι τιμές της προβολής της σε κάθε άξονα είναι περισσότερο πολυάριθμες από ό,τι για ένα κανονικό πλέγμα[14].

Οι ακολουθίες χαμηλής απόκλισης χρησιμοποιούνται πολύ σε προσομοιώσεις Monte Carlo. Μια προσομοίωση Monte Carlo χρησιμοποιείται για τη μοντελοποίηση της πιθανότητας διαφορετικών αποτελεσμάτων σε μια διαδικασία που δεν μπορεί εύκολα να προβλεφθεί λόγω της παρέμβασης τυχαίων μεταβλητών. Είναι μια τεχνική που χρησιμοποιείται για την κατανόηση του αντίκτυπου του ρίσκου και της αβεβαιότητας[15].

Υπάρχουν πολλοί αλγόριθμοι για τη δημιουργία ακολουθίων χαμηλής απόκλισης.

Συνήθεις ακολουθίες χαμηλής απόκλισης είναι οι ακολουθίες Van der Corput, Halton, Sobol και Hammersley.

3.4.1 Ακολουθία Van der Corput

Η ακολουθία Van der Corput είναι η απλούστερη μονοδιάστατη ακολουθία χαμηλής απόκλισης. Για κάθε μη αρνητικό αριθμό k μπορεί να γραφτεί χρησιμοποιώντας ως βάση έναν πρώτο αριθμό p ως:

$$k = a_0 + a_1 p + a_2 p^2 + \cdots + a_r p^r$$

όπου κάθε a_i είναι ένας ακέραιος αριθμός στο διάστημα $[0, p]$. Μπορούμε να ορίσουμε τη συνάρτηση Φ_p του k ως:

$$\Phi_p(k) = a_0/p + a_1/p^2 + a_2/p^2 + \cdots + a_r/p^{r+1}$$

. Για παράδειγμα, έστω $p = 2$ η ακολουθία που προκύπτει από τη συνάρτηση $\Phi_2(k)$ για $k = 0, 1, 2, \cdots$ ονομάζεται ακολουθία Van der Corput[16].

Υλοποίηση αλγορίθμου ακολουθίας Van der Corput

Στον Αλγόριθμο 3 βλέπουμε τον κώδικα που επιτυγχάνει τη διαδικασία που εξηγήθηκε προηγουμένως για έναν αριθμό n με βάση p. Οι τιμές που επιστρέφει είναι μεταξύ του 0 και 1.

3 Βελτιστοποίηση υπερπαραμέτρων με την ακολουθία Van der Corput

```
while n do
    denom = denom*base
    n, remainder = divmod(n, base)
    vdc += remainder/float(denom)
end while
```

3.4.2 Ακολουθία Halton

Η ακολουθία Halton είναι η πιο βασική ακολουθία χαμηλής απόκλισης για πολλαπλές διαστάσεις, η οποία μπορεί να θεωρηθεί ως το δομικό στοιχείο άλλων ακο-

λουθιών χαμηλής απόκλισης. Η ακολουθία Halton είναι μια γενική ακολουθία διαστάσεων s στον υπερκύβο $[0,1]^s$. Η πρώτη διάσταση στην ακολουθία Halton είναι με βάση 2 της ακολουθίας Van der Corput και η δεύτερη διάσταση είναι με βάση 3 της ακολουθίας Van der Corput. Η διάσταση s της ακολουθίας Halton είναι η ακολουθία Van der Corput χρησιμοποιώντας διαφορετικό πρώτο αριθμό $p1, p2, \cdots, ps$ ως βάση σε κάθε διάσταση. Καθώς γίνεται η βάση της ακολουθίας Van der Corput μεγαλύτερή όσο αυξάνονται οι διαστάσεις, χρειάζεται όλο και περισσότερος χρόνος για να γεμίσει ο υπερκύβος (για παράδειγμα, στους 25ος και 26ος οι πρώτοι αριθμοί είναι ο 97 και 101 αντίστοιχα). Η ακολουθία αντιστοιχεί τον πρώτο αριθμό p και έχει κύκλους μήκους p με αριθμούς μονότονα αυξανόμενους. Αυτό το χαρακτηριστικό κάνει τους αρχικούς όρους δύο ακολουθιών να συσχετίζονται σε μεγάλο βαθμό, τουλάχιστον με τον πρώτο κύκλο κάθε ακολουθίας[17].

Μπορούμε να ορίσουμε την ακολουθία Halton ως εξής:

Έστω d οι διαστάσεις του χώρου δειγματοληψίας. Οποιαδήποτε ακολουθία πρώτων αριθμών $p1, p2, \cdots, pd$ ορίζει μια ακολουθία συναρτήσεων $\Phi_{p1}, \Phi_{p1}, \cdots, \Phi_{pd}$, όπου αντιστοιχούν για k όρους και d διαστάσεων, τα σημεία Halton είναι:

$$\left(\Phi_{p1}(k), \Phi_{p2}(k), \cdots, \Phi_{pd}(k)\right), k = 0, 1, 2, \cdots, n-1$$

όπου $p1 < p2 < \cdots < pd - 1$ και n είναι ο συνολικός αριθμός σημείων Halton σε κάθε διάσταση[16].

Στο πρόβλημα της διάστασης s, κάθε διάσταση της ακολουθίας Halton γίνεται με διαφορετικό πρώτο αριθμό βάσης p. Κάθε φορά που ο αριθμός των ψηφίων στο n αυξάνεται κατά μία θέση, Το κλάσμα του n γίνεται γίνεται συντελεστής του p με αποτέλεσμα να δημιουργείται ένα λεπτότερο πλέγμα. Έτσι, σε κάθε βήμα καθώς το n αυξάνει, τα σημεία της ακολουθίας Halton γεμίζουν όλο και καλύτερα τα καρτεσιανά πλέγματα[17].

Υλοποίηση αλγορίθμου

Όπως περιγράφηκε και προηγουμένως, ο αλγόριθμος για την ακολουθία Van der Corput καλείται επαναληπτικά με διαφορετική βάση κάθε φορά. Επίσης. όπως και στους προηγούμενους αλγορίθμους έχουμε την επιλογή να ορίσουμε τον τύπο κάθε παραμέτρου. Στον Αλγόριθμο 4 μπορούμε να δούμε τον ενδεικτικό αλγόριθμο για

τη συγκεκριμένη υλοποίηση.

4 Βελτιστοποίηση υπερπαραμέτρων με την ακολουθία Halton

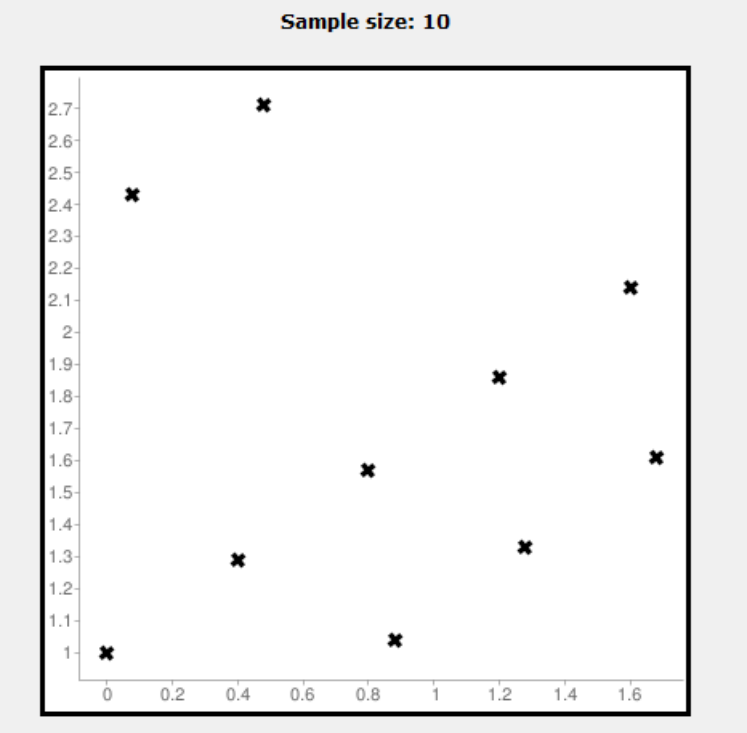
```
for i = 0, i < Number_of_parameters, i+ = 1 do
    for j = 0, j < Number_of_Trials, j+ = 1 do
        Lines[i][j]=vdc(j,base)
    end for
    next(base)
end for
combinations=Take_combinations(Lines)
for i = 0, i < Number_of_parameters, i+ = 1 do
    combinations[i]=Adjust(l_bound[j],u_bound[j])
    if type[i] == integer then
        combinations[i]=round(combinations[i])
    end if
end for
for i = 0, i < Number_of_Trials, i+ = 1) do
    Current_Result= BlackBox(combinations[i])
    if CurrentResult < Best_solution then
        Best_solution=Current_Result
        Best_parameters=combinations[i]
    end if
end for
```

- **Number_of_Trials**: Αριθμός δοκιμών.
- **Current_Result**: Αποτέλεσμα τους προβλήματος μετά την εκτέλεση του.
- **Best_solution**: Η καλύτερη λύση που βρέθηκε.
- **Best_parameters**: Οι παράμετροι που οδήγησαν στην καλύτερη λύση.
- **Lines**: Όλα τα δείγματα κάθε παραμέτρου.
- **l_bound**,**u_bound**: Τα όρια κάθε παραμέτρου.
- **type**: Είδος παραμέτρου (ακέραιος ή πραγματικός).

- **Number_of_parameters**: Το πλήθος των παραμέτρων (διαστάσεις του προβλήματος).
- **BlackBox()**: Το πρόβλημα ως προς επίλυση.
- **combinations**: Όλοι οι συνδυασμοί των παραμέτρων.
- **Take_combinations()**: Δημιουργεί όλους τους συνδυασμούς.
- **vdc()**: Αλγόριθμος Van Der Corput,
- **next()**: Επιλογή επόμενου πρώτου αριθμού.
- **Adjust()**: Προσαρμογή τιμών Halton με βάση τα όρια των παραμέτρων.

Ο κάθε συνδυασμός είναι της μορφής: Ο πρώτος συνδυασμούς θα είναι το πρώτο στοιχείο της Halton ακολουθίας της κάθε παραμέτρου, ο δεύτερος συνδυασμός θα είναι το δεύτερο στοιχείο της Halton ακολουθίας της κάθε παραμέτρου και ούτω καθεξής. Επίσης, ο αλγόριθμος επιστρέφει τιμές μεταξύ του 0 και 1 και για αυτό μετά τη δημιουργία των ακολουθιών, οι τιμές προσαρμόζονται αντίστοιχα για κάθε παράμετρο.

Παράδειγμα εκτέλεσης αλγορίθμου



Σχήμα 3.4: Συνδυασμοί παραμέτρων αλγορίθμου Halton

Στο Σχήμα 3.4 βλέπουμε τα σημεία που δημιουργήθηκαν στην εκτέλεση του αλγορίθμου Halton.

Απεικόνιση 3.3: Παράδειγμα αλγορίθμου Halton

```
{
"f(0.0,1.0) = 1.0 Current best solution 1.0"
"f(0.4,1.29) = 1.41 Current best solution 1.0"
"f(0.8,1.57) = 1.98 Current best solution 1.0"
"f(1.2,1.86) = 2.63 Current best solution 1.0"
"f(1.6,2.14) = 3.34 Current best solution 1.0"
"f(0.08,2.43) = 2.43 Current best solution 1.0"
"f(0.48,2.71) = 2.8 Current best solution 1.0"
"f(0.88,1.04) = 1.78 Current best solution 1.0"
"f(1.28,1.33) = 2.56 Current best solution 1.0"
"f(1.68,1.61) = 3.36 Current best solution 1.0"
```

```
Best Parameters: x=0,y=1
Best solution: 1
}
```

Όπως βλέπουμε στην Απεικόνιση 3.3, ο αλγόριθμος για την ακολουθία Halton θα πάρει τους συνδυασμούς που δημιούργησε και θα τους δοκιμάσει. Σε κάθε δοκιμή κρατάμε την τιμή της συνάρτησης και κάνουμε σύγκριση με την καλύτερη λύση που έχει βρεθεί έως τώρα. Στο τέλος αφού ολοκληρωθούν οι δοκιμές μπορούμε να δούμε το αποτέλεσμα καθώς και τις παραμέτρους που μας οδήγησαν σε αυτό. Όπως και στους προηγούμενους αλγορίθμους ο τρόπος λειτουργίας του αλγορίθμου για την ακολουθία Halton μπορεί να μεταφερθεί και σε προβλήματα περισσότερων διαστάσεων.

3.4.3 Ακολουθία Hammersly

Η ακολουθία Hammersley είναι ίδια με την ακολουθία Halton εκτός από την πρώτη διάσταση όπου τα σημεία βρίσκονται σε ίση απόσταση μεταξύ τους. Δηλαδή η πρώτη διάσταση υπολογίζεται με k/n όπου n ο αριθμός δειγμάτων και $k = \{0, 1, \cdots, n-1\}$. Οι επόμενες διαστάσεις υπολογίζονται με την ακολουθία Van der Corput με τον ίδιο τρόπο που υπολογίζονται στην ακολουθία Halton.

Όπως και στην ακολουθία Halton με τον ίδιο τρόπο μπορούμε να ορίσουμε την ακολουθία Hammersly:

Έστω d οι διαστάσεις του χώρου δειγματοληψίας. Οποιαδήποτε ακολουθία πρώτων αριθμών $p1, p2, \cdots, pd-1$ ορίζει μια ακολουθία συναρτήσεων $\Phi_{p1}, \Phi_{p2}, \cdots, \Phi_{pd-1}$, όπου αντιστοιχούν για k όρους και d διαστάσεων, τα σημεία Hammersly είναι:

$$(k/n, \Phi_{p1}(k), \Phi_{p2}(k), \cdots, \Phi_{pd-1}(k)), k = 0, 1, 2, \cdots, n-1$$

όπου $p1 < p2 < \cdots < pd-1$ και n είναι ο συνολικός αριθμός σημείων Hammersly σε κάθε διάσταση.

Υλοποίηση αλγορίθμου

Όπως περιγράφηκε και προηγουμένως ο Hammersly αλγόριθμος απλώς είναι ο αλγόριθμος Van der Corput που καλείται επαναληπτικά με διαφορετική βάση κάθε

φορά, εκτός της πρώτης διάστασης που τα σημεία ισαπέχουν μεταξύ τους. Άρα πολύ απλά μπορούμε να υλοποιήσουμε τον αλγόριθμο για την ακολουθία Hammersly όπως φαίνεται στον Αλγόριθμο 5.

5 Βελτιστοποίηση υπερπαραμέτρων με την ακολουθία Hammersly

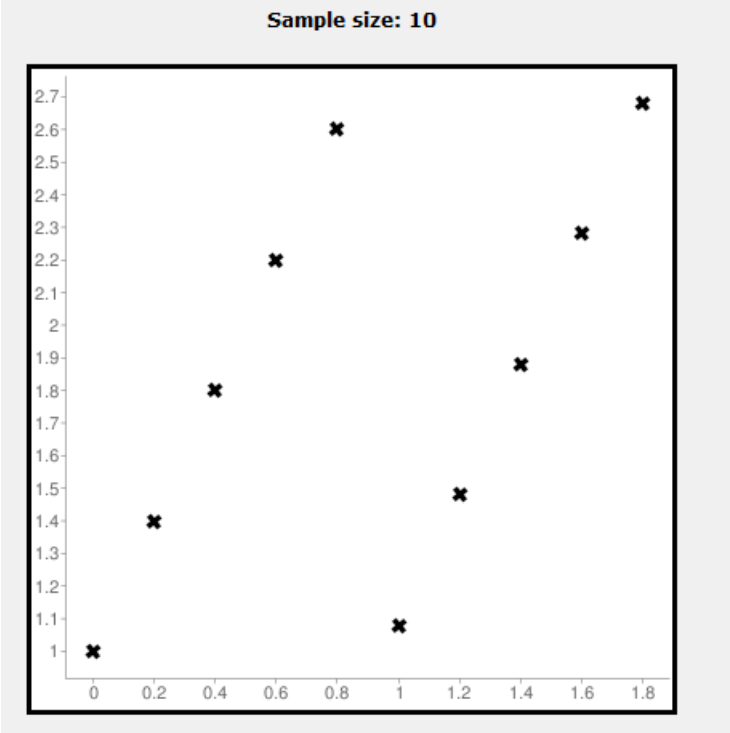
```
for i = 0, i < Number_of_Trials, i+ = 1 do
    Lines[0][i]=i/Number_of_Trials
end for
for i = 1, i < Number_of_parameters, i+ = 1 do
    for j = 0, j < Number_of_Trials, j+ = 1 do
        Lines[i][j]=vdc(j,base)
    end for
    next(base)
end for
combinations=Take_combinations(Lines)
for i = 0, i < Number_of_parameters, i+ = 1 do
    combinations[i]=Adjust(l_bound[j],u_bound[j])
    if type[i] == integer then
        combinations[i]=round(combinations[i])
    end if
end for
for i = 0, i < Number_of_Trials, i+ = 1) do
    Current_Result= BlackBox(combinations[i])
    if CurrentResult < Best_solution then
        Best_solution=Current_Result
        Best_parameters=combinations[i]
    end if
end for
```

- **Number_of_Trials**: Αριθμός δοκιμών.
- **Current_Result**: Αποτέλεσμα τους προβλήματος μετά την εκτέλεση του.
- **Best_solution**: Η καλύτερη λύση που βρέθηκε.

- **Best_parameters**: Οι παράμετροι που οδήγησαν στην καλύτερη λύση.
- **Lines**: Όλα τα δείγματα κάθε παραμέτρου.
- **l_bound**,**u_bound**: Τα όρια κάθε παραμέτρου.
- **type**: Είδος παραμέτρου (ακέραιος ή πραγματικός).
- **Number_of_parameters**: Το πλήθος των παραμέτρων (διαστάσεις του προβλήματος).
- **BlackBox()**: Το πρόβλημα ως προς επίλυση.
- **combinations**: Όλοι οι συνδυασμοί των παραμέτρων.
- **Take_combinations()**: Δημιουργεί όλους τους συνδυασμούς.
- **vdc()**: Αλγόριθμος Van Der Corput,
- **next()**: Επιλογή επόμενου πρώτου αριθμού.
- **Adjust()**: Προσαρμογή τιμών Hammersly με βάση τα όρια των παραμέτρων.

Ο κάθε συνδυασμός είναι της μορφής: Ο πρώτος συνδυασμούς θα είναι το πρώτο στοιχείο της Hammersly ακολουθίας της κάθε παραμέτρου, ο δεύτερος συνδυασμός θα είναι το δεύτερο στοιχείο της Hammersly ακολουθίας της κάθε παραμέτρου και ούτω καθεξής. Επίσης, ο αλγόριθμος επιστρέφει τιμές μεταξύ του 0 και 1 και για αυτό μετά τη δημιουργία των ακολουθιών, οι τιμές προσαρμόζονται αντίστοιχα για κάθε παράμετρο.

Παράδειγμα εκτέλεσης αλγορίθμου



Σχήμα 3.5: Συνδυασμοί παραμέτρων αλγορίθμου Hammersly

Στο Σχήμα 3.5 βλέπουμε τα σημεία που δημιουργήθηκαν στην εκτέλεση του αλγορίθμου Hammersly.

Απεικόνιση 3.4: Παράδειγμα αλγορίθμου Hammersly

```
"f(0.0,1.0) = 1.0 Current best solution 1.0"
"f(0.2,1.4) = 1.43 Current best solution 1.0"
"f(0.4,1.8) = 1.89 Current best solution 1.0"
"f(0.6,2.2) = 2.36 Current best solution 1.0"
"f(0.8,2.6) = 2.85 Current best solution 1.0"
"f(1.0,1.08) = 2.01 Current best solution 1.0"
"f(1.2,1.48) = 2.45 Current best solution 1.0"
"f(1.4,1.88) = 2.92 Current best solution 1.0"
"f(1.6,2.28) = 3.4 Current best solution 1.0"
"f(1.8,2.68) = 3.89 Current best solution 1.0"

Best Parameters: x=0.0,y=1.0
```

Best solution: 1

Όπως βλέπουμε στην Απεικόνιση 3.4, ο αλγόριθμος για την ακολουθία Hammerly θα πάρει τους συνδυασμούς που δημιούργησε και θα τους δοκιμάσει. Σε κάθε δοκιμή κρατάμε την τιμή της συνάρτησης και κάνουμε σύγκριση με την καλύτερη λύση που έχει βρεθεί έως τώρα. Στο τέλος αφού ολοκληρωθούν οι δοκιμές μπορούμε να δούμε το αποτέλεσμα καθώς και τις παραμέτρους που μας οδήγησαν σε αυτό. Όπως και στους προηγούμενους αλγορίθμους ο τρόπος λειτουργίας του αλγορίθμου για την ακολουθία Hammerly μπορεί να μεταφερθεί και σε προβλήματα περισσότερων διαστάσεων.

3.4.4 Ακολουθία Sobol

Αρχικά, ας υποθέσουμε ότι εργαζόμαστε σε μία μόνο διάσταση. Στόχος μας είναι λοιπόν να δημιουργηθεί μια ακολουθία τιμών $x^1, x^2, \cdots, 0 < x^i < 1$, με χαμηλή απόκλιση. Χρειαζόμαστε πρώτα ένα σύνολο αριθμών κατεύθυνσης(direction numbers) $u_1, u_2, \cdots$. Κάθε u_i είναι ένα δυαδικό κλάσμα το οποίο μπορεί να γραφτεί ως

$$u_i = m_i/2^i$$

όπου m_i είναι ένας περιττός ακέραιος αριθμός, $0 < m_i < 2^i$. Για να υπολογίσουμε τα u_i, ξεκινάμε επιλέγοντας ένα πολυώνυμο με συντελεστές που είναι 0 ή 1, το πολυώνυμο αυτό ονομάζεται πρωτόγονο πολυώνυμο (primitive polynomial). Έτσι, μπορούμε να επιλέξουμε, ας πούμε,

$$P = x^d + a + 1x^{d-1} + \cdots + a_{d-1}x + 1$$

όπου κάθε a_i είναι 0 ή 1 και το P είναι ένα πρωτόγονο πολυώνυμο βαθμού d (αν το P είναι πρωτόγονο πολυώνυμο τότε ο σταθερός όρος, είναι απαραίτητα ίσος με 1). Εφόσον το P είναι ένα πρωτόγονο πολυώνυμο, η επιλογή του πολυωνύμου μπορεί να γίνει αυθαίρετα. Αφού επιλέξουμε ένα πολυώνυμο, χρησιμοποιούμε τους συντελεστές του για να ορίσουμε έναν αναδρομικό τύπο για τον υπολογισμό του u_i έτσι,

$$u_i = a_1 u_{i-1} \oplus a_2 u_{i-2} \oplus \cdots \oplus a_{d-1} u_{i-d+1} \oplus u_{i-d} \oplus u_{i-d}/2^d, i > d$$

όπου $\oplus$ υποδηλώνει τη λογική πράξη xor bit με bit και ο τελευταίος όρος είναι ο u_{i-d} μετατοπισμένος δεξιά d θέσεις. Ισοδύναμα, μπορούμε να εκφράσουμε αναδρομικό τύπο για τον υπολογισμό του m_i

$$m_i = 2a_1 n_{i-1} \oplus 2^2 a_2 m_{i-2} \oplus \cdots \oplus 2^{d-1} a_{d-1} m_{i-d+1} \oplus 2^d m_{i-d} \oplus m_{i-d}$$

. Χρησιμοποιώντας ένα πρωτόγονο πολυώνυμο βαθμού d, οι τιμές των $m_1, m_2 \cdots, m_d$ μπορούν να επιλεχθούν ελεύθερα με την προϋπόθεση ότι κάθε m_i είναι περιττός και $m_i < 2^i$, οι επόμενες τιμές $m_{d+1}, m_{d+2}, \cdots$ υπολογίζονται από τον αναδρομικό τύπο. Έστω το πρωτόγονο πολυώνυμο

$$x^3 + x + 1$$

βαθμού 3. Επομένως ο αναδρομικός τύπος που προκύπτει είναι

$$m_i = 4m_{i-2} \oplus 8m_{i-3} \oplus m_{i-3}, (a_1 = 0, a_2 = 1)$$

και έστω οι αρχικές τιμές είναι $m_1 = 1, m_2 = 3, m_3 = 7$. Τότε

$$m_4 = 12 \oplus 8 \oplus 1 = 5$$

$$m_5 = 28 \oplus 24 \oplus 3 = 7$$

και ούτω καθεξής. Οι αριθμοί κατεύθυνσης υπολογίζονται με τον τύπο $u_i = m_i/2^i$ άρα: Αν $m_1 = 1, m_2 = 3, m_3 = 7, m_4 = 5, m_5 = 7$ τότε οι αριθμοί κατεύθυνσης είναι $v_1 = 0.5, v_2 = 0.75, v_3 = 0.875, v_4 = 0.3125, v_5 = 0.21875$. Τέλος, για τη δημιουργία της ακολουθία $x^1, x^2, \cdots$ υπολογίζουμε

$$x^n = b_1 u_1 \oplus b_2 u_2 \oplus \cdots$$

όπου $\cdots b_3 b_2 b_1$ είναι η δυαδική αναπαράσταση του n. Αυτή είναι αρχική μέθοδος υπολογισμού της ακολουθίας Sobol [18].

Χρήση γκρι κώδικα (Gray code)

Ο γκρι κώδικας για τον αριθμό n ορίζεται ως

$$G(n) = n \oplus [n/2]$$

[19]. Για παράδειγμα, χρησιμοποιώντας τον παραπάνω κώδικα παίρνουμε

$$G(1) = 1, G(2) = 3, G(3) = 2, G(4) = 6, G(5) = 7$$

και και ούτω καθεξής. Επίσης, ο γκρι κώδικας έχει την ιδιότητα ότι η δυαδική αναπαράσταση του G(n) και G(n-1) διαφέρουν μόνο κατά μια θέση, ο δείκτης του πρώτου 0 ψηφίου από τα δεξιά στη δυαδική αναπαράσταση του n-1.

Με αυτόν τον τρόπο μπορούμε να δημιουργήσουμε την ακολουθία Sobol ως εξής:

$$x^n = g_1 u_1 \oplus g_2 u_2 \oplus g_3 u_3 \cdots$$

όπου $\cdots g_3 g_2 g_1$ είναι η δυαδική αναπαράσταση του $G(n)$

Πιο γρήγορος τρόπος δημιουργίας της ακολουθίας Sobol

Ένας πιο γρήγορος τρόπος για τη δημιουργία της ακολουθίας προτάθηκε από τον Antono και Saleev με την προσθήκη του γκρίζου κώδικα (Gray code). Έστω μια ακολουθία $\{c_i\}$, όπου c_i είναι ο δείκτης του πρώτου 0 ψηφίου από τα δεξιά στη δυαδική αναπαράσταση του $i = (\cdots i_3 i_2 i_1)2$. Έχουμε $c_0 = 1, c_1 = 2, c_2 = 1, c_3 = 3, c_4 = 1, c_5 = 2$ και ούτω καθεξής. Έτσι μπορούμε να δημιουργήσουμε την ακολουθία Sobol θεωρώντας $x^0 = 0$ ως

$$x^{n+1} = x^n u_{cn}$$

Για τη δημιουργία ακολουθίας s διαστάσεων, μπορούμε να χρησιμοποιήσουμε την ίδια διαδικασία επιλέγοντας για κάθε διάσταση ένα διαφορετικό πρωτόγονο πολυώνυμο.

Υλοποίηση αλγορίθμου

Στη συγκεκριμένη υλοποίηση, ο αλγόριθμος για την ακολουθία Sobol έχει βασιστεί πάνω στον τρόπο τον οποίο προτάθηκε από τους Antono και Saleev με την προσθήκη του γκρι κώδικα.

6 Βελτιστοποίηση υπερπαραμέτρων με την ακολουθία Sobol

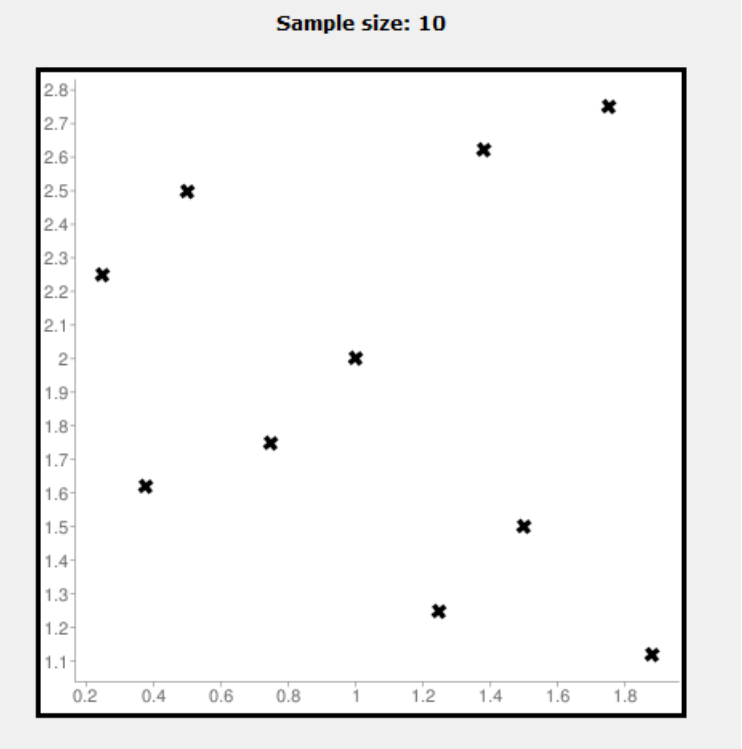
```
Initialize(v[40:30])
for i = 0, i < Number_of_parameters, i+ = 1 do
    New_Poly=Choose_Poly()
    Degree= Poly_Degree(New_Poly)
    coefficients=coef(New_Poly)
    BraFox(v,Degree,coefficients)
end for
quasi=0
for i = 0, i < Number_of_Trials, i+ = 1 do
    for j = 0, j < Number_of_parameters, j+ = 1 do
        Lines[j] =xor(quasi[j],v[j,RightZero(Number_of_Trials)])
    end for
end for
combinations=Take_combinations(Lines)
for i = 0, i < Number_of_parameters, i+ = 1 do
    combinations[i]=Adjust(l_bound[j],u_bound[j])
    if type[i] = integer then
        combinations[i]=round(combinations[i])
    end if
end for
for i = 0, i < Number_of_Trials, i+ = 1 do
    Current_Result= BlackBox(combinations[i])
    if CurrentResult < Best_solution then
        Best_solution=Current_Result
        Best_parameters=combinations[i]
    end if
end for
```

- **Number_of_Trials**: Αριθμός δοκιμών.
- **Current_Result**: Αποτέλεσμα τους προβλήματος μετά την εκτέλεση του.
- **Best_solution**: Η καλύτερη λύση που βρέθηκε.

- **Best_parameters**: Οι παράμετροι που οδήγησαν στην καλύτερη λύση.
- **Lines**:: Όλα τα δείγματα κάθε παραμέτρου.
- **l_bound**,**u_bound**: Τα όρια κάθε παραμέτρου.
- **type**: Είδος παραμέτρου (ακέραιος ή πραγματικός).
- **Number_of_parameters**: Το πλήθος των παραμέτρων (διαστάσεις του προβλήματος).
- **BlackBox()**: Το πρόβλημα ως προς επίλυση.
- **combinations**: Όλοι οι συνδυασμοί των παραμέτρων.
- **Take_combinations()**: Δημιουργεί όλους τους συνδυασμούς.
- **Adjust()**: Προσαρμογή τιμών Hammersly με βάση τα όρια των παραμέτρων.
- **Initialize(v)**: Δημιουργία πίνακα με τιμές m(όπως στη θεωρία) για έως και 40 διαστάσεις
- **Choose_Poly()**: Επιλογή νέου πολυωνύμου.
- **Poly_Degree**: Εύρεση του βαθμού πολυωνύμου.
- **coef()**: Εύρεση των συντελεστών πολυωνύμου.
- **BraFox()**: Υπολογισμός των v όπως περιγράφεται στο Bratley και Fox [18],στο κεφάλαιο 2.
- **RightZero()**: Εύρεση της θέσης του πρώτου μηδενικού από δεξιά σε έναν αριθμό στη δυαδική του μορφή.

Ο κάθε συνδυασμός είναι της μορφής: Ο πρώτος συνδυασμούς θα είναι το πρώτο στοιχείο του Sobol ακολουθίας της κάθε παραμέτρου, ο δεύτερος συνδυασμός θα είναι το δεύτερο στοιχείο της Sobol ακολουθίας της κάθε παραμέτρου και ούτω καθεξής. Επίσης, ο αλγόριθμος επιστρέφει τιμές μεταξύ του 0 και 1 και για αυτό μετά τη δημιουργία των ακολουθιών, οι τιμές προσαρμόζονται αντίστοιχα για κάθε παράμετρο.

Παράδειγμα εκτέλεσης αλγορίθμου



Σχήμα 3.6: Συνδυασμοί παραμέτρων αλγορίθμου Sobol

Στο Σχήμα 3.6 βλέπουμε τα σημεία που δημιουργήθηκαν στην εκτέλεση του αλγορίθμου για την ακολουθία Sobol.

Απεικόνιση 3.5: Παράδειγμα εκτέλεσης του αλγορίθμου Sobol

```
{
"f(1.0,2.0) = 2.5 Current best solution 2.5"
"f(1.5,1.5) = 3.0 Current best solution 2.5"
"f(0.5,2.5) = 2.6 Current best solution 2.5"
"f(0.75,1.75) = 2.07 Current best solution 2.07"
"f(1.75,2.75) = 3.86 Current best solution 2.07"
"f(1.25,1.25) = 2.5 Current best solution 2.07"
"f(0.25,2.25) = 2.28 Current best solution 2.07"
"f(0.38,1.62) = 1.71 Current best solution 1.71"
"f(1.38,2.62) = 3.35 Current best solution 1.71"
"f(1.88,1.12) = 4.25 Current best solution 1.71"
```

```
Best Parameters: x=0.38,y=1.62
Best solution: 1.71
}
```

Όπως βλέπουμε στην Απεικόνιση 3.5, ο αλγόριθμος για την ακολουθία Sobol θα πάρει τους συνδυασμούς που δημιούργησε και θα τους δοκιμάσει. Σε κάθε δοκιμή κρατάμε την τιμή της συνάρτησης και κάνουμε σύγκριση με την καλύτερη λύση που έχει βρεθεί έως τώρα. Στο τέλος αφού ολοκληρωθούν οι δοκιμές μπορούμε να δούμε το αποτέλεσμα καθώς και τις παραμέτρους που μας οδήγησαν σε αυτό. Όπως και στους προηγούμενους αλγορίθμους, ο τρόπος λειτουργίας του αλγορίθμου για την ακολουθία Sobol μπορεί να μεταφερθεί και σε προβλήματα περισσότερων διαστάσεων.

3.5 Λατινικός υπερκύβος (Latin hyperCube)

Ο Latin Hypercube Sampling (LHS) είναι μια μέθοδος δημιουργίας τυχαίων δειγμάτων. Χρησιμοποιείται ευρέως στην προσομοίωση Monte Carlo, επειδή μπορεί να μειώσει δραστικά τον αριθμό των δοκιμών που απαιτούνται για να επιτευχθεί ένα ικανοποιητικό αποτέλεσμα αποτέλεσμα. Ο LHS βασίζεται στο λατινικό τετράγωνο (latin square) σχέδιο, το οποίο έχει ένα μόνο δείγμα σε κάθε γραμμή και στήλη. Ένας «υπερκύβος» (hypercube) είναι ένας κύβος με περισσότερες από τρεις διαστάσεις[20].

Η μέθοδος πίσω από τη δειγματοληψία LHS: Έστω $X \in (0, 1)$ τυχαία μεταβλητή Η μονοδιάστατη δειγματοληψία του latin hypercube για τη μεταβλητή X περιλαμβάνει τη διαίρεση του διαστήματος $(0, 1)$ σε n ίσα υπόδιαστηματα και στη συνέχεια επιλέγοντας μια τυχαία τιμή σε κάθε υποδιάστημα. Ως απλό παράδειγμα, ας υποθέσουμε ότι χρειαζόσασταν 4 τυχαίες τιμές της μεταβλητής X. Αρχικά, χωρίζουμε το διάστημα $(0, 1)$ σε ίσα διαστήματα δηλαδή $(0, 0.25), (0.25, 0.5), (0.5, 0, 75), (0.75, 1)$. Στη συνέχεια σε κάθε διάστημα επιλέγεται τυχαία μια τιμή, δίνοντάς μας 4 διαφορετικά δείγματα. Ο LHS δύο διαστάσεων δεν είναι πολύ πιο περίπλοκο. Υποθέτοντας ότι οι δύο μεταβλητές μας, x1 και x2 είναι ανεξάρτητες, ακολουθούμε τη μονοδιάστατη μέθοδο για να βρούμε μονοδιάστατα δείγματα για x1 και x2 χωριστά. Μόλις έχουμε δύο λίστες δειγμάτων, τα συνδυάζουμε, τυχαία, σε ζεύγη. Για δειγματοληψία

n-διαστάσεων απλώς χρησιμοποιείται η ίδια μέθοδος[20].

Υλοποίηση αλγορίθμου

Η υλοποίηση του αλγορίθμου LHS έγινε ως εξής: Όπως περιγράφτηκε προηγουμένως για κάθε παράμετρο ξεχωριστά χωρίζουμε το διάστημα της υποδιαστήματα. Όσες είναι οι δοκιμές που θέλουμε τόσα είναι και τα υποδιαστήματα. Αφού δημιουργηθούν τα υποδιαστήματα της κάθε παραμέτρου,παίρνουμε από κάθε υποδιάστημα τυχαία μια τιμή. Στο τέλος αφού έχουν δημιουργηθεί όλα τα δείγματα κάθε παραμέτρου οι συνδυασμοί μεταξύ τους γίνονται τυχαία. Επίσης όπως και στις προηγούμενες υλοποιήσεις αλγορίθμων έχουμε τη δυνατότητα να επιλέξουμε τον τύπο της κάθε παραμέτρου. Σε περίπτωση ακέραιας παραμέτρου κατά τη δειγματοληψία μπορεί να ''παραβιαστεί'' ο κανόνας του LHS, δηλαδή να έχουμε παραπάνω από ένα σημεία στην ίδια στήλη ή γραμμή. Αυτό όμως δεν είναι πρόβλημα καθώς ο αλγόριθμος θέλουμε να μπορεί να δουλεύει και για ακέραιες παραμέτρους επομένως είναι κάτι που δεν μπορούμε να αποφύγουμε. Στον Αλγόριθμο 7 βλέπουμε το ενδεικτικό αλγόριθμο.

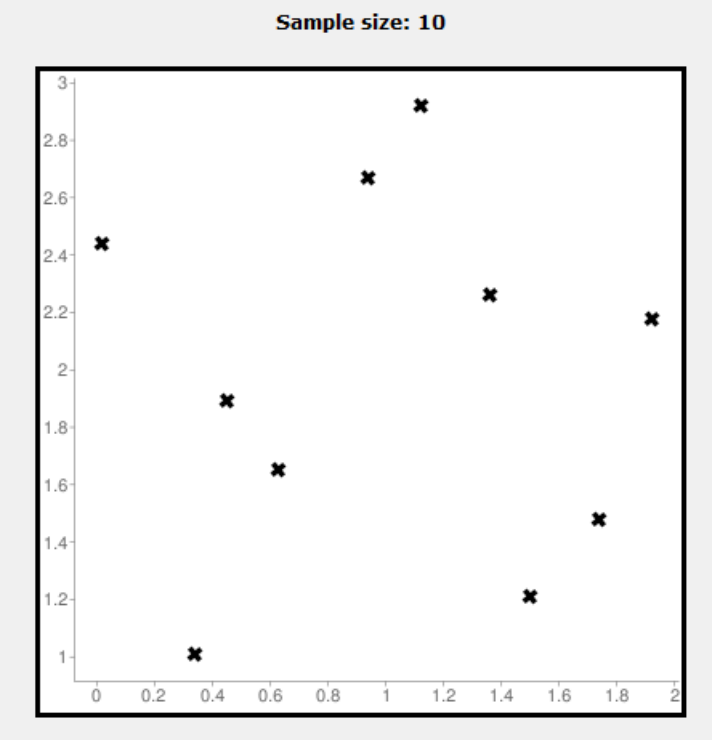
7 Βελτιστοποίηση υπερπαραμέτρων αλγορίθμου λατινικού υπερκύβου

for $i = 0, i < Number\_of\_parameters, i+ = 1$ **do**
  Slices.add(SliceBounds(l_bound[i],u_bound[i],$Number\_of\_Trials$))
  for $j = 0, j < Number\_of\_Trials - 1, j+ = 1$ **do**
    temp.add(randomgenarator(Slices[0][j],Slices[0][j+1]))
    Lines.add(temp)
  end for
end for
combinations=Random_combinations(Lines)
for $i = 0, i < Number\_of\_parameters, i+ = 1$ **do**
  if $type[i] = integer$ **then**
    combinations[i]=round(combinations[i])
  end if
end for
for $i = 0, i < Number\_of\_Trials, i+ = 1)$ **do**
  Current_Result= BlackBox(combinations[i])
  if $CurrentResult < Best\_solution$ **then**
    Best_solution=Current_Result
    Best_parameters=combination[i]
  end if
end for

- **Number_of_Trials**: Αριθμός δοκιμών.
- **Current_Result**: Αποτέλεσμα τους προβλήματος μετά την εκτέλεση του.
- **Best_solution**: Η καλύτερη λύση που βρέθηκε.
- **Best_parameters**: Οι παράμετροι που οδήγησαν στην καλύτερη λύση.
- **Lines**: Όλα τα δείγματα κάθε παραμέτρου.
- **l_bound**,**u_bound**: Τα όρια κάθε παραμέτρου.
- **type**: Είδος παραμέτρου (ακέραιος ή πραγματικός).

- **Number_of_parameters**: Το πλήθος των παραμέτρων (διαστάσεις του προβλήματος).
- **BlackBox()**: Το πρόβλημα ως προς επίλυση.
- **combinations**: Όλοι οι συνδυασμοί των παραμέτρων.
- **Random_combinations()**: Δημιουργεί όλους τους συνδυασμούς τυχαία.
- **SliceBounds()**: Χωρισμός του διαστήματος της παραμέτρου σε υποδιαστήματα.
- **randomgenarator()**: Δημιουργεί μια τυχαία τιμή με βάση τα όρια της κάθε παραμέτρου.

3.5.1 Παράδειγμα αλγορίθμου LHS



Σχήμα 3.7: Συνδυασμοί παραμέτρων αλγορίθμου λατινικού υπερκύβου

Στο Σχήμα 3.7 βλέπουμε τα σημεία που δημιουργήθηκαν στην εκτέλεση του αλγορίθμου LHS.

Απεικόνιση 3.6: Παράδειγμα αλγορίθμου Λατινικός υπερκύβος

```
"f(0.34,1.01) = 1.13 Current best solution 1.13"
"f(1.5,1.21) = 3.08 Current best solution 1.13"
"f(1.74,1.48) = 3.52 Current best solution 1.13"
"f(1.36,2.26) = 3.08 Current best solution 1.13"
"f(1.92,2.18) = 3.87 Current best solution 1.13"
"f(0.63,1.65) = 1.89 Current best solution 1.13"
"f(0.94,2.67) = 3.0 Current best solution 1.13"
"f(0.45,1.89) = 1.99 Current best solution 1.13"
"f(0.02,2.44) = 2.44 Current best solution 1.13"
"f(1.12,2.92) = 3.36 Current best solution 1.13"

Best Parameters: x=0.34,y=1.01
Best solution: 1.13
```

Όπως βλέπουμε στην Απεικόνιση 3.6, ο αλγόριθμος LHS θα πάρει τους συνδυασμούς που δημιούργησε και θα τους δοκιμάσει. Σε κάθε δοκιμή κρατάμε την τιμή της συνάρτησης και κάνουμε σύγκριση με την καλύτερη λύση που έχει βρεθεί έως τώρα. Στο τέλος αφού ολοκληρωθούν οι δοκιμές μπορούμε να δούμε το αποτέλεσμα καθώς και τις παραμέτρους που μας οδήγησαν σε αυτό. Όπως και στους προηγούμενους αλγορίθμους ο τρόπος λειτουργίας του αλγορίθμου LHS μπορεί να μεταφερθεί και σε προβλήματα περισσότερων διαστάσεων.

3.6 Συστήματα λογισμικού εύρεσης βέλτιστων υπερπαραμέτρων

Σε αυτή την ενότητα θα δούμε τα λογισμικά Dakota και Nomad που χρησιμοποιούνται για την επίλυση προβλημάτων βελτιστοποίησης υπερπαραμέτρων μαύρου κουτιού. Και στις δύο περιπτώσεις ο αλγόριθμος που χρησιμοποιούν είναι μία παραλλαγή του αλγόριθμου MADS (Mesh Adaptive Direct Search).

3.6.1 Βασικά στοιχεία του αλγορίθμου MADS

Ο αλγόριθμος Mesh Adaptive Direct Search (MADS) χρησιμοποιείται για βελτιστοποίηση προβλημάτων μαύρου κουτιού. Ο MADS είναι σχεδιασμένος για τέτοια προβλήματα και στοχεύει στην καλύτερη δυνατή λύση με μικρό αριθμό αξιολογήσεων[21].

Αυτή η μέθοδος δημιουργεί επαναλήψεις σε μια σειρά πλεγμάτων με ποικίλο μέγεθος. Ένα πλέγμα είναι μια διακριτοποίηση του χώρου των παραμέτρων. Ωστόσο, όπως υποδηλώνει και το όνομά του, ο αλγόριθμος εκτελεί μια προσαρμοστική αναζήτηση στα πλέγματα συμπεριλαμβανομένου του ελέγχου της βελτίωσης των πλεγμάτων.

Ο στόχος κάθε επανάληψης του αλγορίθμου MADS είναι η δημιουργία ενός δοκιμαστικού σημείου στο πλέγμα που βελτιώνει την τρέχουσα καλύτερη λύση. Όταν μια επανάληψη αποτυγχάνει να το επιτύχει αυτό, η επόμενη επανάληψη ξεκινά σε ένα λεπτότερο πλέγμα. Κάθε επανάληψη αποτελείται από δύο κύρια βήματα που ονομάζονται ”Search step” και ”Poll steps”[22]. Το ”Search step” είναι κρίσιμο στην πράξη, επειδή είναι τόσο ευέλικτο και μπορεί να βελτιώσει σημαντικά την απόδοση. Το ”Search step” περιορίζεται από τη θεωρία να επιστρέφει σημεία στο υποκείμενο πλέγμα, αλλά φυσικά, προσπαθεί να εντοπίσει ένα σημείο που βελτιώνει την τρέχουσα καλύτερη λύση.

Το ”Poll step” ορίζεται πιο αυστηρά, αν και εξακολουθεί να υπάρχει κάποια ευελιξία στον τρόπο με τον οποίο εφαρμόζεται. Το ”Poll step” δημιουργεί δοκιμαστικά σημεία πλέγματος κοντά στην καλύτερη τρέχουσα λύση. Δεδομένου ότι το βήμα ”Poll step” είναι η βάση της ανάλυσης σύγκλισης, είναι το μέρος του αλγορίθμου όπου έχει συγκεντρωθεί η περισσότερη έρευνα. Ενδεικτικός αλγόριθμος 2.

8 Βελτιστοποίηση υπερπαραμέτρων με τον αλγόριθμο MADS

Αρχικοποίηση: $x_0 \in \mathbb{R}^n$ μα είναι το αρχικό σημείο και ο μετρητής επαναλήψεων $k = 0$

Κύριος βρόγχος

repeat

SEARCH στο πλέγμα για την εύρεση καλύτερης λύσης απο το x_k

if SEARCH αποτύχει **then**

POLL στο πλέγμα για την εύρεση καλύτερης λύσης από το x_k

end if

if SEARCH καλύτερη λύση βρέθηκε από το x_k είτε από το SEARCH είτε από το POLL **then**

Κάλεσε το x_{k+1} και το συρρίκνωσε το πλέγμα.

else

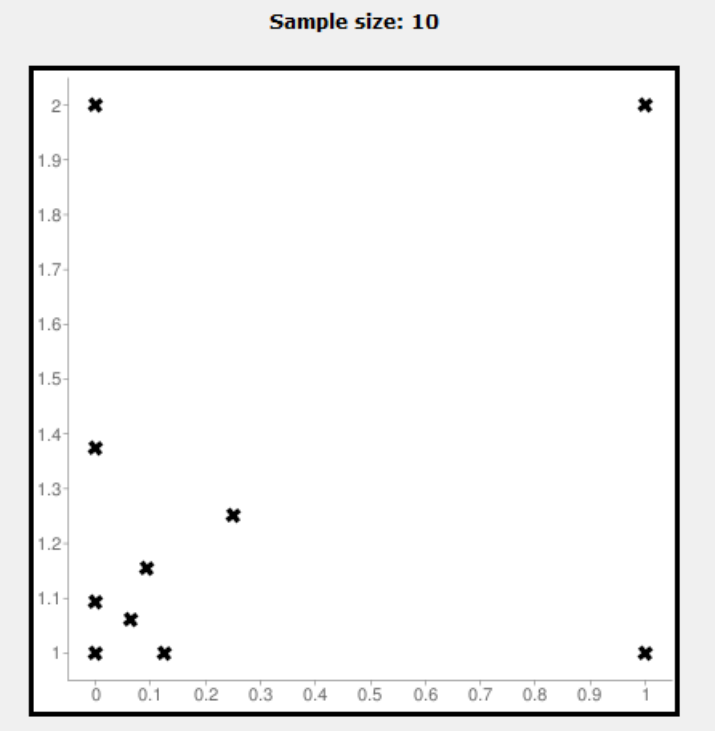
Θέσε $x_{k+1} = x_k$ και λέπτυνε το πλέγμα.

end if

Ενημέρωσε τις παραμέτρους και $k = k + 1$

until Τα κριτήρια να ικανοποιηθούν

3.6.2 Παράδειγμα εκτέλεσης με το λογισμικό Dakota



Σχήμα 3.8: Συνδυασμοί παραμέτρων λογισμικού Dakota MADS

Στο Σχήμα 3.8 παρατηρούμε κατά την εκτέλεση του λογισμικού ότι οι περισσότεροι συνδυασμοί που έχει επιλέξει είναι "μαζεμένοι" σε ένα σημείο. Αυτό οφείλεται όπως αναφέραμε και προηγουμένως στο ότι ο αλγόριθμος MADS σε κάθε εκτέλεση προσπαθεί όσο το δυνατόν καλύτερα να βρει μέσα το πλέγμα παραμέτρους που θα δώσουν καλύτερη λύση.

Απεικόνιση 3.7: Παράδειγμα λογισμικού Dakota με MADS

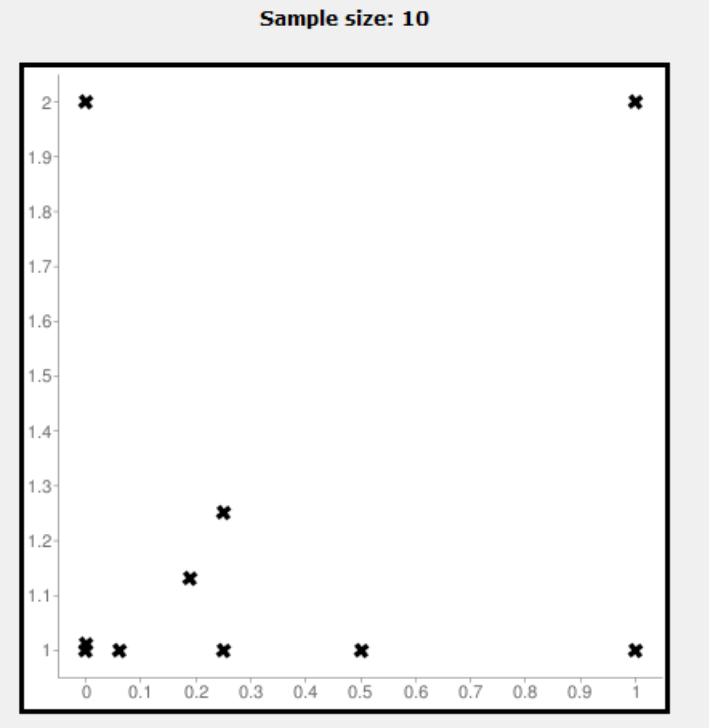
```
"f(1,2) = 2.5 Current best solution 2.5 "
"f(0,2) = 2 Current best solution 2"
"f(0,1) = 1 Current best solution 1"
"f(1,1) = 2 Current best solution 1"
"f(0.25,1.25) = 1.3  Current best solution 1"
"f(0,1.375) = 1.0625 Current best solution 1"
"f(0.125,1) = 1.16195 Current best solution 1"
"f(0.0625,1.0625 ) = 1.066 Current best solution 1"
"f(0.09375,1.15625) = 1.16385138  Current best solution 1"
```

”f(0,1.09375) = 1.09375 Current best solution 1”

Best Parameters: x=0.0,y=1.0
Best solution: 1

Στην Απεικόνιση 3.7 βλέπουμε την εκτέλεση της συνάρτησης f για 10 δοκιμές με το λογισμικό Dakota.

3.6.3 Παράδειγμα εκτέλεσης με το λογισμικό Nomad



Σχήμα 3.9: Συνδυασμοί παραμέτρων λογισμικού Nomad

Στο Σχήμα 3.9 παρατηρούμε παρόμοια συμπεριφορά με το λογισμικό Nomad, κάτι τι οποίο είναι λογικό καθώς στο βασικό του πυρήνα χρησιμοποιεί τον αλγόριθμο MADS.

Απεικόνιση 3.8: Παράδειγμα λογισμικού Nomad με MADS

”f(1,2) = 2.5 Current best solution 2.5 ”
”f(0,2) = 2 Current best solution 2”
”f(0,1) = 1 Current best solution 1”
”f(1,1) = 2 Current best solution 1”

```
"f(0.25,1.25) = 1.3  Current best solution 1"
"f(0.25,1) = 1.0625 Current best solution 1"
"f(0.19,1.13) = 1.16195 Current best solution 1"
"f(0.06,1) = 1.0036 Current best solution 1"
"f(0.5,1) = 1.25 Current best solution 1"
"f(0,1.01) = 1.01 Current best solution 1"

Best Parameters: x=0.0,y=1.0
Best solution: 1
```

Κεφάλαιο 4

Υπολογιστική μελέτη

Σε αυτό το κεφάλαιο θα παρουσιαστούν τα αποτελέσματα των αλγορίθμων που αναλύθηκαν στο προηγούμενο κεφάλαιο και θα διατυπωθούν οι παρατηρήσεις και η διαδικασία που χρησιμοποιήθηκε για την εκτέλεση της μελέτης.

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε για την υλοποίηση των αλγορίθμων και την εκτέλεση της υπολογιστικής μελέτης είναι η Python έκδοσης 3.10 με περιβάλλον ανάπτυξης Visual Studio Code έκδοσης 1.71. Για τη δημιουργία του προβλήματος μαύρου κουτιού έγινε η χρήση της γλώσσας προγραμματισμού C++. Το λογισμικό όπου και έγινε η βελτιστοποίηση υπερπαραμέτρων είναι ο λύτης γραμμικού προγραμματισμού Gurobi.

4.1 Πρόγραμμα μαύρο κουτί

Τα συστήματα λογισμικού Nomad και Dakota για τη χρήση τους απαιτούν το καθένα αρχείο εισόδου όπου δίνονται οι πληροφορίες για τη φύση του προβλήματος όπως καθώς και ένα εκτελέσιμο αρχείο που το διαχειρίζονται ως το πρόβλημα μαύρου κουτιού και περιέχει μέσα το λογισμικό προς βελτιστοποίηση. Ο τρόπος λειτουργίας τους είναι ο εξής: Πρώτα δημιουργούν το συνδυασμό των υπερπαραμέτρων σε ένα αρχείο εξόδου, οι πληροφορίες για τις υπερπαραμέτρους βρίσκονται όλες στο αρχείο εισόδου. Τα συστήματα λογισμικού μετά διαβάζουν τα αρχεία εξόδου και δοκιμάζουν τους συνδυασμούς στο εκτελέσιμο αρχείο που το διαχειρίζονται ως μαύρο κουτί όπως αναφέρθηκε προηγούμενος. Το πρόβλημα που υπάρχει είναι ότι το κάθε λογισμικό έχει τις δικές του ιδιαιτερότητες στον τρόπο συγγραφής των αρχείων εισόδων, εξόδων καθώς και του εκτελέσιμου. Για αυτό το λόγο υλοποιήθηκε στη γλώσσα προγραμματισμού ένα εκτελέσιμο αρχείο το όποιο ο σκοπός του

είναι να δουλεύει σε όποια περίπτωση τύχει είτε είναι το λογισμικό Nomad είτε το λογισμικό Dakota. Ο τρόπος λειτουργίας των αλγορίθμων βασίστηκαν σε αυτή τη λογική, δηλαδή αρχεία εισόδου και εξόδου. Επίσης για τα αρχεία εισόδου έχουν υλοποιηθεί κώδικες όπου με την τροποποίηση ενός κοινού αρχείου τροποποιούνται τα υπόλοιπα αυτοματοποιημένα. Παρακάτω βλέπουμε τον τρόπο λειτουργίας του εκτελέσιμου αρχείου σε πολύ απλή μορφή.

9 Εκτελέσιμο αρχείο

```
read(problem name)
if Dakota as sampler then
    hyperparameters=read(Dakota.ouput)
    time=Gurobi(hyperparameters,problem name)
    return time
else if Nomad as sampler then
    hyperparameters=read(Nomad.ouput)
    time=Gurobi(hyperparameters,problem name)
    return time
else if Algorithm as sampler then
    hyperparameters=read(Algorithm.ouput)
    time=Gurobi(hyperparameters,problem name)
    return time
end if
```

4.1.1 Λύτης γραμμικού προγραμματισμού Gurobi

Ο λύτης γραμμικού προγραμματισμού Gurobi είναι ένα λογισμικό που επιλύει διάφορων τύπων προβλημάτων βελτιστοποίηση. Στη συγκεκριμένη υπολογιστική μελέτη χρησιμοποιήθηκαν προβλήματα της μορφής MPS. Η μορφή MPS είναι από της πιο παλιές και διαδεδομένες μορφές για την αποθήκευση μαθηματικών μοντέλων για προβλήματα γραμμικού προγραμματισμού και μικτού ακέραιου προγραμματισμού. Ο γραμμικός προγραμματισμός (Linear Programming) ή αλλιώς γραμμική βελτιστοποίηση, είναι μέθοδος για την επίτευξη του καλύτερου αποτελέσματος (για παράδειγμα: μέγιστο κέρδος ή ελάχιστο κόστος) σε ένα μαθηματικό υπόδειγμα, του οποίου οι προϋποθέσεις (περιορισμοί) είναι ένα σύνολο γραμμικών σχέσεων

των μεταβλητών του. Ο μικτός ακέραιος προγραμματισμός είναι παρόμοιος με τον γραμμικό προγραμματισμό με τη διαφορά ότι προϋποθέτει ορισμένες αλλά όχι όλες οι μεταβλητές να λαμβάνουν μόνο ακέραιες τιμές.

Ο λύτης γραμμικού προγραμματισμού Gurobi χρησιμοποιεί προηγμένους αλγορίθμους όπως LP αλγόριθμοι, QP αλγόριθμοι και MIP αλγόριθμοι για την επίλυση των μαθηματικών προβλημάτων. Το λογισμικό βρίσκει τη λύση επομένως και επιστρέφει το χρόνο που έκανε για τη βρει. Ο στόχος της συγκεκριμένης μελέτης είναι η εύρεση υπερπαραμέτρων τέτοιων ώστε για το δεδομένο πρόβλημα να το λύσουμε σε λιγότερο χρόνο σε σχέση με το χρόνο που έκανε να επιλυθεί με της προεπιλεγμένες υπερπαραμέτρους του λογισμικού.

Υπερπαράμετροι λύτη Gurobi

Το συγκεκριμένο λογισμικό περιέχει πολλές διαφορετικές υπερπαράμετρους. Για αυτήν τη μελέτη επιλέχθηκαν 10 από αυτές. Από τις 10 υπερπαραμέτρους οι δύο είναι τύπου πραγματικές και οι υπόλοιπες 8 είναι τύπου ακέραιες. Κάθε υπερπαράμετρος έχει ένα συγκεκριμένο εύρος τιμών που μπορεί να πάρει καθώς και μια προκαθορισμένη τιμή. Στον Πίνακα 4.1 αναγράφονται οι υπερπαράμετροι που επιλέχθηκαν για τη συγκεκριμένη υπολογιστική μελέτη. Οι προκαθορισμένες τιμές επιλέγονται από το λογισμικό Gurobi κατά την επίλυση ενός προβλήματος όταν δεν παρεμβαίνουμε καθόλου εμείς.

Πίνακας 4.1: Υπερπαράμετροι λύτη Gurobi

Υπερπαράμετρος	Μικρότερη τιμή	Μεγαλύτερη τιμή	Προκαθορισμένη τιμή	Τύπος
markowitztol	0.0001	0.999	0.0078125	Πραγματικός
perturbvalue	0	Άπειρη τιμή	0.0002	Πραγματικός
normadjust	-1	4	-1	Ακέραιος
method	-1	5	-1	Ακέραιος
cutaggpasses	-1	2^{31}-1	-1	Ακέραιος
mircuts	-1	2	-1	Ακέραιος
cutpasses	-1	2^{31}-1	-1	Ακέραιος
rins	-1	2^{31}-1	-1	Ακέραιος
gomorypasses	-1	2^{31}-1	-1	Ακέραιος
simplexpricing	-1	3	-1	Ακέραιος

4.1.2 Διαδικασία εκτέλεσης της μελέτης

Από τη βιβλιοθήκη μεικτού ακεραίου προγραμματισμού (MIPLIB 2017) επιλέχθηκε ο συμπιεσμένος φάκελος benchmark όπου περιέχει 240 προβλήματα τύπου MPS. Από τα 240 προβλήματα επιλέχθηκαν 65 προβλήματα με κριτήριο το χρόνο που χρειάζονται για να φτάσουν στη λύση με την προεπιλεγμένες τιμές του λογισμικού Gurobi, που ορίσαμε τα 100 δευτερόλεπτα. Κάθε αλγόριθμος και λογισμικό είχε συνολικά 100 δοκιμές να πραγματοποιήσει για κάθε πρόβλημα. Πριν την εκκίνηση της διαδικασίας της βελτιστοποίησης το πρόβλημα εκτελείται με τις προεπιλεγμένες τιμές. Για κάθε δοκιμή υπήρχε ένα χρονικό όριο 3600 δευτερολέπτων. Το χρονικό όριο προσδέθηκε για να αποφύγουμε περιπτώσεις όπου το πρόβλημα μπορεί να πάρει πολλές ώρες για να λυθεί. Στον Πίνακα 4.1 είδαμε τις υπερπαραμέτρους που επιλέχθηκαν καθώς και τις αντίστοιχες πληροφορίες. Για τη συγκεκριμένη μελέτη επιλέχθηκε διαφορετικό εύρος τιμών για κάποιες υπερπαραμέτρους στο Πίνακα 4.2 βλέπουμε τα νέο εύρος τιμών κάθε παραμέτρου.

Πίνακας 4.2: Τιμές υπερπαραμέτρων λύτη Gurobi

Υπερπαράμετρος	Μικρότερη τιμή	Μεγαλύτερη τιμή	Προκαθορισμένη τιμή	Τύπος
markowitztol	0.0001	0.999	0.0078125	Πραγματικός
perturbvalue	0	0.01	0.0002	Πραγματικός
normadjust	-1	4	-1	Ακέραιος
method	-1	3	-1	Ακέραιος
cutaggpasses	-1	10	-1	Ακέραιος
mircuts	-1	2	-1	Ακέραιος
cutpasses	-1	10	-1	Ακέραιος
rins	-1	10	-1	Ακέραιος
gomorypasses	-1	3	-1	Ακέραιος
simplexpricing	-1	3	-1	Ακέραιος

Ο Πίνακας 4.3 περιέχει γενικές πληροφορίες για τα προβλήματα όπως το σύνολο και τους τύπους των μεταβλητών καθώς και τη λύση τους.

Πίνακας 4.3: Πληροφορίες προβλημάτων

Προβλήματα	Σύνολο Μεταβλητών	Περιορισμοί	Δυαδικές Μεταβλητές	Ακέραιες Μεταβλητές	Πραγματικές Μεταβλητές	Λύση
30n20b8	18,380	576	18,318	62	0	302
CMS750_4	11,697	16,381	7,196	0	4,501	252
air04	8,904	823	8,904	0	0	56,137
air05	7,195	426	7,195	0	0	26,374
app1-1	2,480	4,926	1,225	1,225	0	-3
blp-ic98	13,640	717	13,550	0	90	4,491.44
cbs-cta	24,793	10,112	2,467	0	22,326	0
comp07-2idx	17,264	21,235	17,155	109	0	6
dano3_3	13,873	3,202	69	0	13,804	576.34
drayage-25-23	11,090	4,630	11,025	0	65	101,282.64
eil33-2	4,516	32	4,516	0	0	934
ex10	17,680	69,608	0	17,680	0	100
ex9	10,404	40,962	0	0	0	81
fast0507	63,009	507	63,009	0	0	174
fastxgemm-n2r6s0t2	784	5,998	48	0	736	230
fiball	34,219	3,707	33,960	1	0	138
hypothyroid-k1	2,602	5,195	2,601	1	0	-2,851

irp	20,315	39	20,315	0	0	12,159.49
istanbul-no-cutoff	5,282	20,346	30	0	5,252	204.08
mcsched	1,747	2,107	1,745	0	2	211,913
mzzv11	10,240	9,499	9,989	251	0	-21,718
mzzv42z	11,717	10,460	11,482	235	0	-20,540
n2seq36q	22,480	2,565	0	0	0	52,200
neos-1122047	5,100	57,791	100	0	5,000	161
neos-1171448	4,914	13,206	2,457	0	2,457	-309
neos-1171737	2,340	4,179	1,170	0	1,170	-195
neos-1445765	20,617	2,147	2,150	0	18,467	-17,783
neos-1582420	10,100	10,180	10,000	100	0	90.99
neos-2746589-doon	50,936	31,530	50,704	224	8	2,008.19
neos-2987310-joes	27,837	29,015	3,051	0/0	24,786	-607,702,988.3
neos-3004026-krka	17,030	12,545	16,900	130	0	0
neos-3988577-wolgan	25,870	44,662	25,870	0	0	-
neos-4413714-turia	190,402	2,303	190,201	0	201	45.37
neos-4722843-widden	77,723	113,555	73,349	20	4,354	25,009.66
neos-4738912-atrato	6,216	1,947	1,120	5,096	0	283,627,956.59
neos-5107597-kakapo	3,114	6,498	2,976	0	138	3,644.99
neos-662469	18,235	1,085	17,907	328	0	184,379.99

neos-860300	1,385	850	1,384	0	1	3,200.99
neos-933966	31,762	12,047	27,982	0	3,780	318
neos-950242	5,760	34,224	5,520	240	0	4
neos-957323	57,756	3,757	57,756	0	0	-237.7566815
neos-960392	59,376	4,744	59,376	0	0	-238
net12	14,115	14,021	1,603	0	12,512	214
netdiversion	129,180	119,589	129,180	0	0	242
ns1208400	2,883	4,289	2,880	0/1	3	2
ns1830653	1,629	2,932	1,458	0	171	20,622
ns1952667	13,264	41	0	13,264	0	0
nu25-pr12	5,868	2,313	5,832	36	0	53,904.99
nurseched-sprint02	10,250	3,522	10,230	20	0	57.99
nw04	87,482	36	87,482	0	0	16,862
pg5_34	2,600	225	100	0	2,500	-14,339.35
piperout-27	11,659	18,442	11,514	121	24	8,123.99
piperout-8	10,399	14,589	10,245	130	24	125,054.99
qap10	4150	1,820	4,150	0	0	339.99
rmatr100-p10	7,359	7,260	100	0	7,259	423
rocl-4-11	6,839	10,883	5192	1,016	631	-6,020,203
seymour1	1,372	4,944	451	0	921	410.76370139

supportcase33	20,203	20,489	20,102	101	0	-345
supportcase7	138,844	6,532	451	14	138,379	-1,132.22
swath1	6,805	884	2,306	0	4,499	379.07
swath3	6,805	884	2,706	0	4,099	397.76
traininstance6	10,218	12,309	4,154	2,056	4,008	28,290
triptim1	30,055	15,706	20,456	9,592	7	22.86
uccase12	62,529	121,161	9,072	0	53,457	11,507.40
unitcal_7	25,755	48,939	2,856	0	22,899	19,635,558.24

4.2 Αποτελέσματα

Το Σχήμα 4.2 δείχνει το ποσοστό των προβλημάτων που λύθηκαν σε σχέση με το χρόνο για κάθε αλγόριθμο και λογισμικό. Συγκρίνοντας τις μεθόδους παρατηρούμε τα εξής για για κάθε μέθοδο:

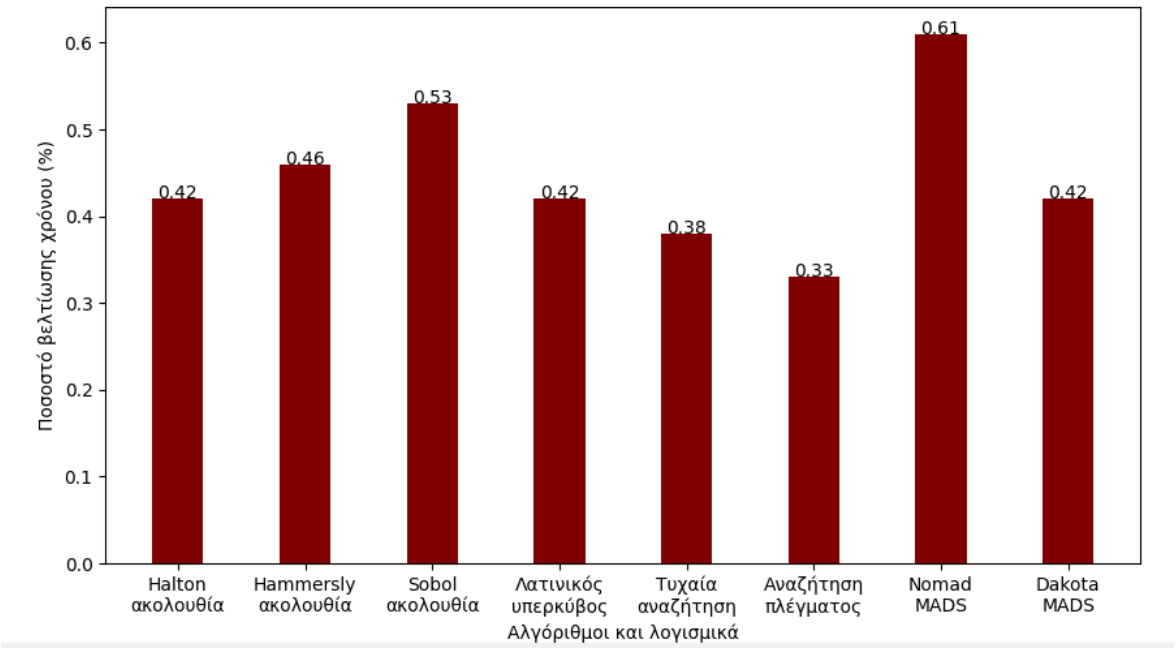
- Χωρίς βελτιστοποίηση το 100% των προβλημάτων λύνεται σε 89 δευτερόλεπτα.
- Για τον αλγόριθμο βασισμένο στην ακολουθία Halton το 100% των προβλημάτων λύνεται σε 52 δευτερόλεπτα.
- Για τον αλγόριθμο βασισμένο στην ακολουθία Hammersly το 100% των προβλημάτων λύνεται σε 48 δευτερόλεπτα.
- Για τον αλγόριθμο βασισμένο στην ακολουθία Sobol το 100% των προβλημάτων λύνεται σε 42 δευτερόλεπτα.
- Για τον αλγόριθμο βασισμένο στον λατινικό υπερκύβο το 100% των προβλημάτων λύνεται σε 52 δευτερόλεπτα.
- Για τον αλγόριθμο βασισμένο στην τυχαία αναζήτηση το 100% των προβλημάτων λύνεται σε 55 δευτερόλεπτα.
- Για τον αλγόριθμο βασισμένο στην αναζήτηση πλέγματος το 100% των προβλημάτων λύνεται σε 60 δευτερόλεπτα.
- Για το λογισμικό Nomad MADS το 100% των προβλημάτων λύνεται σε 35 δευτερόλεπτα.
- Για το λογισμικό Dakota MADS το 100% των προβλημάτων λύνεται σε 52 δευτερόλεπτα.

Μπορούμε να διαπιστώσουμε ότι σε κάθε περίπτωση όλοι οι αλγόριθμοι και τα συστήματα λογισμικού κατάφεραν να μειώσουν το χρόνο για το ίδιο πλήθος προβλημάτων σε σχέση με την περίπτωση χωρίς βελτιστοποίηση. Ο αλγόριθμος αναζήτησης πλέγματος φαίνεται να έχει τη χειρότερη απόδοση ενώ την καλύτερη απόδοση την έχει το λογισμικό Nomad με αρκετή διαφορά από όλες τις υπόλοιπες μεθόδους.

Όπως γνωρίζουμε, το λογισμικό Nomad είναι βασισμένο στον αλγόριθμο MADS. Το ίδιο όμως και το λογισμικό Dakota, όμως παρατηρούμε ότι το λογισμικό Dakota δεν απόδωσε σαν το λογισμικό Nomad που αυτό μπορεί να οφείλεται στις διαφορετικές τροποποιήσεις που έχει προσθέσει το κάθε λογισμικό στο αλγόριθμο MADS. Στη συγκεκριμένη περίπτωση με τα συγκεκριμένα προβλήματα η υλοποίηση του αλγορίθμου MADS από το λογισμικό Nomad φαίνεται να είναι καλύτερη.

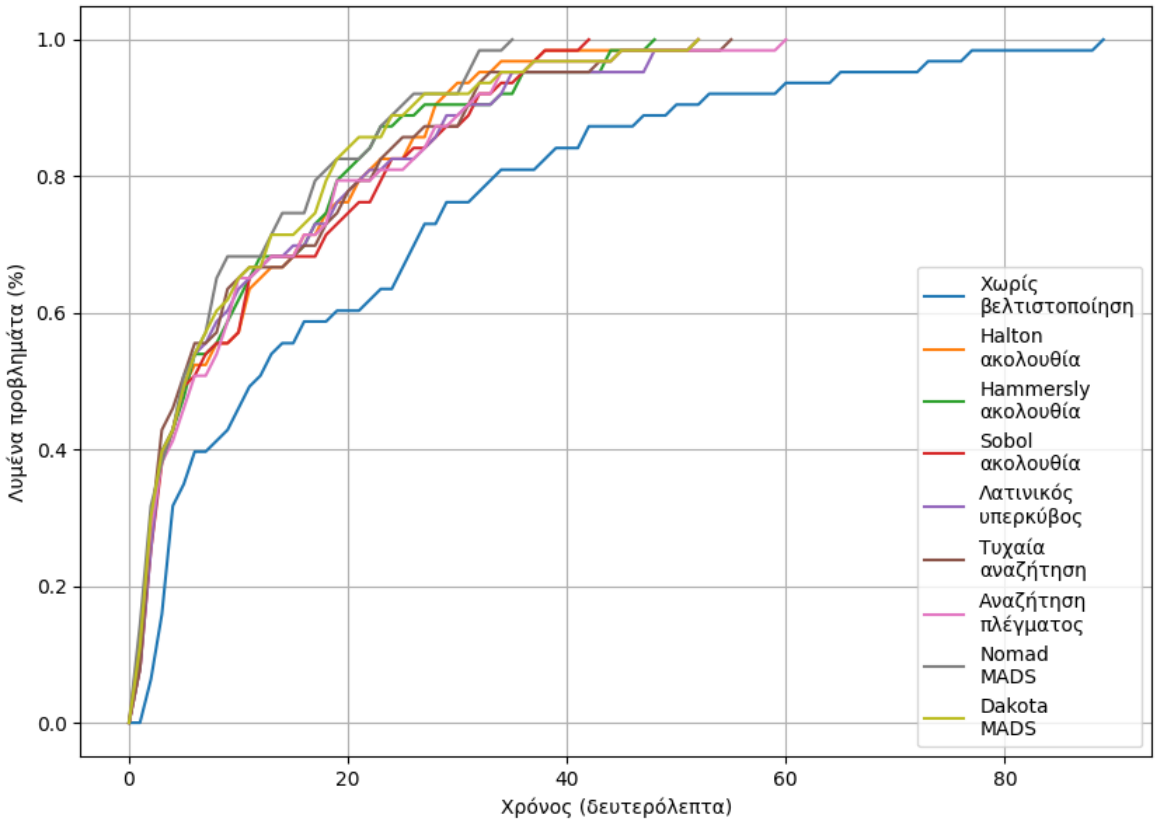
Για τις ακολουθίες χαμηλής απόκλισης, δηλαδή οι Halton, Hammersly και Sobol, ο αλγόριθμος βασισμένος στην ακολουθία Sobol είχε την καλύτερη απόδοση ενώ ο αλγόριθμος βασισμένος στην ακολουθία Halton είχε τη χειρότερη. Η ακολουθία Hammersly βρίσκεται αρκετά κοντά με την ακολουθία Halton και αυτό οφείλεται στο ότι η μόνη διαφορά τους είναι στον τρόπο δημιουργίας της πρώτης διάστασης, αλλά παρατηρούμε ότι με αυτή τη διαφορά πέτυχε καλύτερη απόδοση από την ακολουθία Halton.

Οι αλγόριθμοι λατινικός υπερκύβος και τυχαίας αναζήτησης εισάγουν τυχαιότητα στη διαδικασία. Ο λατινικός υπερκύβος είχε καλύτερη απόδοση σε σχέση με τον αλγόριθμο τυχαίας αναζήτησης. Συγκρίνοντας τους με τον αλγόριθμο αναζήτησης πλέγματος μπορούμε να συμπεράνουμε ότι η τυχαιότητα των αλγορίθμων μας έδωσε καλύτερη απόδοση από το απλώς να δοκιμάζαμε συνδυασμούς με τη μέθοδο του αλγορίθμου πλέγματος.



Σχήμα 4.1: Ποσοστό βελτίωσης συνολικού χρόνου όλων των προβλημάτων

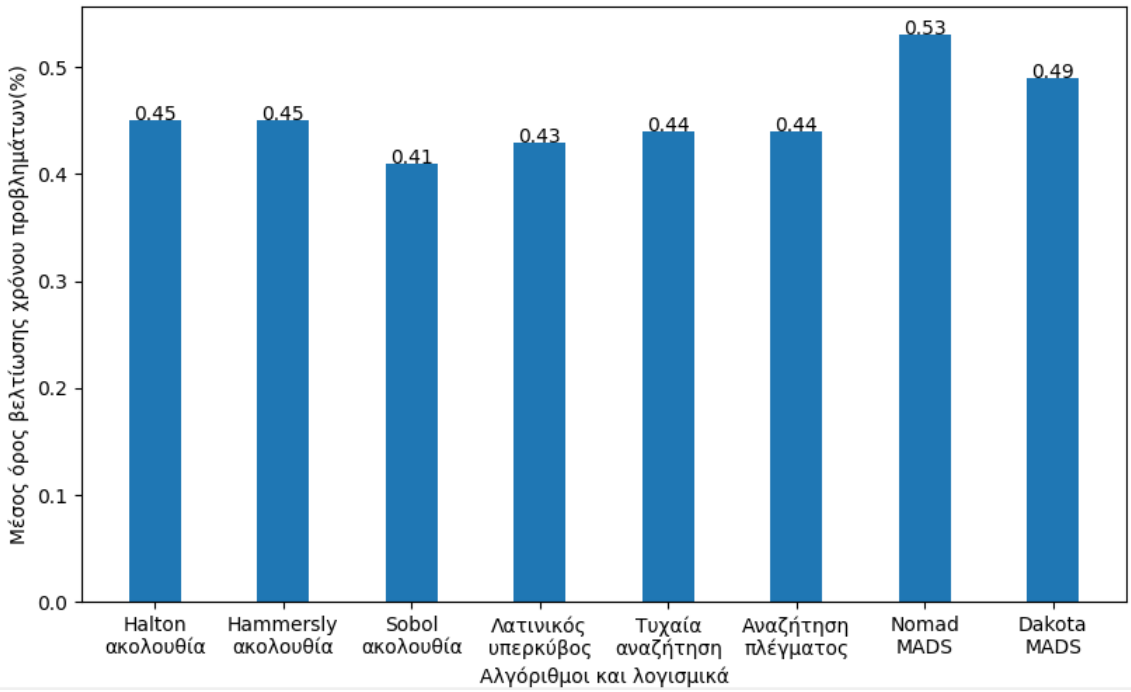
Στο Σχήμα 4.1 βλέπουμε το ποσοστό βελτίωσης του χρόνου που χρειάζονται να λυθούν τα προβλήματα σε σχέση με το χρόνο των προβλημάτων που χρειάζονται για να λυθούν όταν δεν πραγματοποιείται βελτιστοποίηση υπερπαραμέτρων στο λογισμικό Gurobi. Στη συγκεκριμένη περίπτωση, το λογισμικό Nomad θα μας προσφέρει την καλύτερη βελτίωση του χρόνου για τη λύση των συγκεκριμένων προβλημάτων. Στο Σχήμα 4.2, όλες οι μέθοδοι στα αρχικά προβλήματα που χρειαζόταν λίγος χρόνος για να επιλυθούν δεν υπάρχουν μεγάλες αποκλίσεις μεταξύ των μεθόδων. Καθώς μετακινούμαστε στο διάγραμμα από δεξιά προς τα αριστερά δηλαδή σε δυσκολότερα προβλήματα τότε οι μέθοδοι αρχίζουν να έχουν κάποιες διαφορές στα ποσοστά ολοκλήρωσης εκτέλεσης των προβλημάτων.



Σχήμα 4.2: Ποσοστό λυμένων προβλημάτων σε σχέση με το χρόνο

Στο Σχήμα 4.3 φαίνονται ο μέσος όρος βελτίωσης χρόνου κάθε μεθόδου για όλα τα προβλήματα. Με βάση τα Σχήματα 4.2 και 4.1 που αναλύθηκαν προηγουμένως, συνδυάζοντας τα δεδομένα τους μπορούμε να συμπεράνουμε ότι τελικά μπορούμε να δούμε ότι όλοι οι μέθοδοι κατά μέσο όρο έχουν πολύ παρόμοια βελτίωση του χρόνου, με τα λογισμικά Nomad και Dakota να έχουν την καλύτερη. Επιστρέφοντας

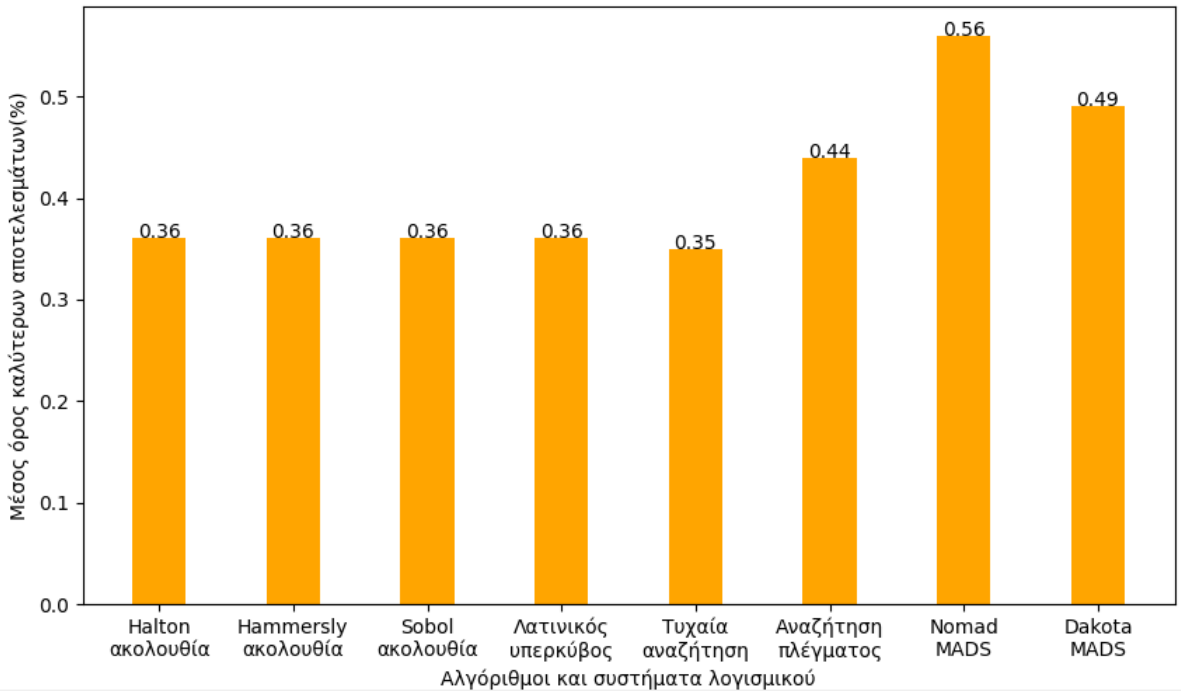
στο Σχήμα 4.2 μπορούμε να δούμε ότι όλοι οι μέθοδοι έχουν παρόμοια γραφική παράσταση με μικρές αποκλίσεις. Μπορούμε τελικά να συμπεράνουμε ότι ο συνολικός χρόνος για την επίλυση όλων των προβλημάτων οφείλεται σε κάποια συγκεκριμένα προβλήματα που μπορεί κάποιος αλγόριθμος ή λογισμικό να έτυχε να είχε χειρότερη απόδοση και έτσι στο σύνολο των προβλημάτων να φαίνεται ότι χρειάζεται περισσότερος χρόνος για να τα λύσει. Βλέποντας όμως το Σχήμα 4.1 παρατηρούμε ότι για παράδειγμα ο αλγόριθμος βασισμένος στο πλέγμα αναζήτησης ενώ φαινόταν ο χειρότερος κατά μέσο όρο έχει αρκετά παρόμοια απόδοση με τους υπόλοιπους αλγορίθμους.



Σχήμα 4.3: Μέσος όρος βελτίωσης χρόνου όλων των προβλημάτων

Όπως αναφέρθηκε και προηγουμένως κάθε αλγόριθμος και κάθε λογισμικό είχε στη διάθεση του να κάνει 100 δοκιμές για κάθε πρόβλημα. Από αυτές τις 100 δοκιμές που μπορούμε να δούμε στο Σχήμα 4.4 κατά μέσο όρο σε ποσοστό, πόσες από αυτές ήταν καλύτερες δηλαδή είχαν καλύτερο χρόνο σε σχέση με το χρόνο της εκτέλεσης του προβλήματος με τις προεπιλεγμένες τιμές του λύτη Gurobi. Παρατηρούμε ότι οι αλγόριθμοι που ήταν βασισμένοι στην ακολουθία Halton, Hammersly και Sobol καθώς και ο λατινικός υπερκύβος και η τυχαία αναζήτηση είχαν σχεδόν ίδια ποσοστά. Ο αλγόριθμος βασισμένος στον αλγόριθμο πλέγματος είχε το καλύτερο ποσοστό από τους προηγούμενους. Αυτό μπορεί να οφείλεται στο ότι για τα

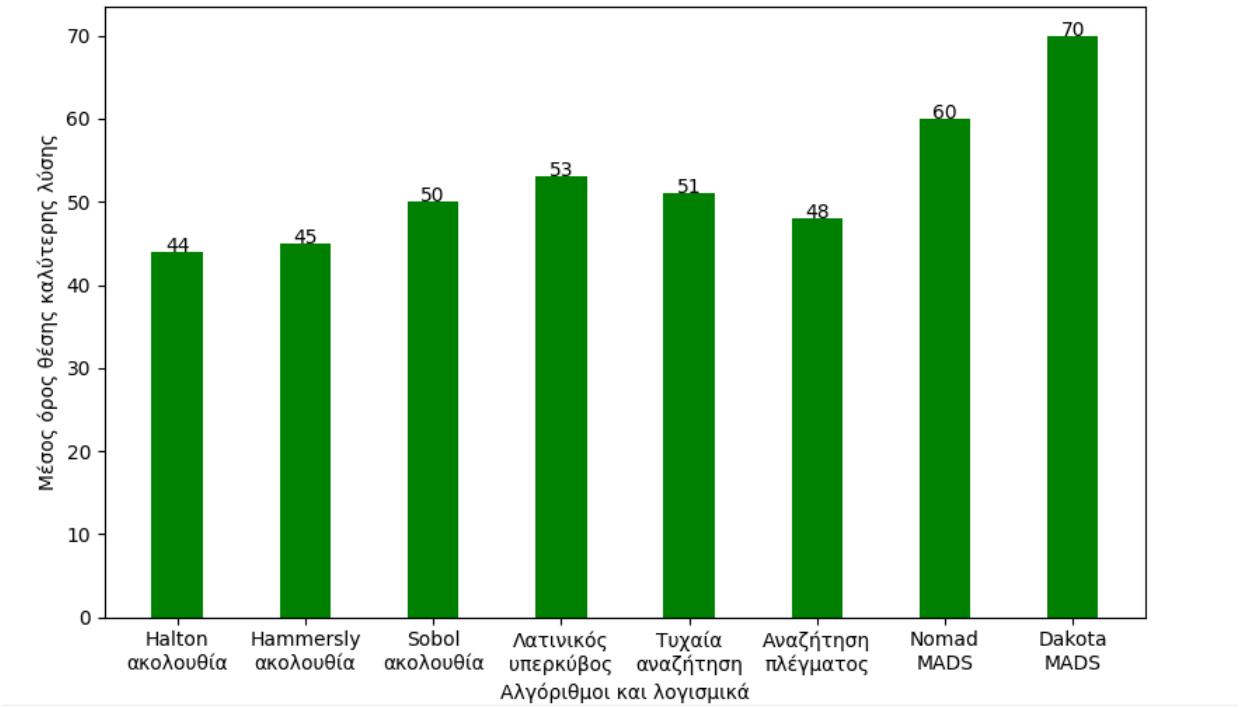
συγκεκριμένα προβλήματα ο αλγόριθμος πλέγματος χώριζε το χώρο των υπερπαραμέτρων σε πιο ικανοποιητικά επίπεδα από τους προηγούμενους. Τα συστήματα λογισμικού Nomad και Dakota είχαν τις καλύτερες αποδόσεις με το Nomad λογισμικό να βρίσκεται στην κορυφή. Όπως έχει αναφερθεί ο αλγόριθμος MADS λαμβάνει υπόψιν του τα προηγούμενα αποτελέσματα και προσπαθεί να επιλέξει από το χώρο των υπερπαραμέτρων, τις καλύτερες υπερπαραμέτρους για την επόμενη δοκιμή που θα του προσφέρουν καλύτερη απόδοση. Επομένως, είναι λογικό και τα δυο συστήματα λογισμικού να έχουν κατά μέσο όρο μεγαλύτερο ποσοστό από τους αλγορίθμους που απλώς εξερευνούν το χώρο των υπερπαραμέτρων "τυφλά" δηλαδή δεν έχουν πληροφορίες για προηγούμενες δοκιμές τους ώστε να προσαρμοστούν ανάλογα.



Σχήμα 4.4: Μέσος όρος εύρεσης καλύτερων αποτελεσμάτων στις 100 δοκιμές ανά πρόβλημα

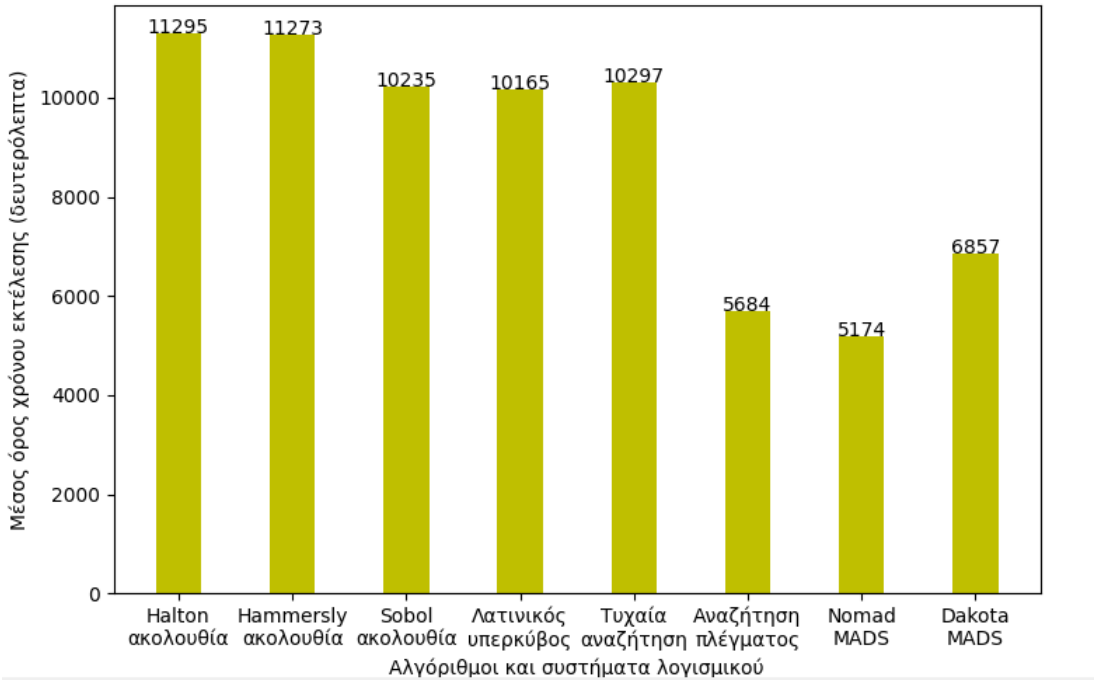
Στο Σχήμα 4.5 βλέπουμε κατά μέσο όρο τη θέση της δοκιμής που βρέθηκε η καλύτερη λύση. Εδώ πάλι μπορούμε να σχολιάσουμε τη συμπεριφορά κυρίως των συστημάτων λογισμικού Dakota και Nomad. Παρατηρούμε ότι και τα δύο λογισμικά κατά μέσο όρο έβρισκαν το καλύτερο αποτέλεσμα μετά από περισσότερες δοκιμές σε σχέση με τους απλούς αλγορίθμους. Είναι λογική η συμπεριφορά αυτή καθώς ο αλγόριθμος MADS με κάθε νέα δοκιμή προσπαθεί να βρει καλύτερο αποτέλεσμα. Για τους υπόλοιπους αλγορίθμους μπορούμε να συμπεράνουμε ότι περίπου στις

μισές δοκιμές έχουν βρει την καλύτερη λύση με τις υπόλοιπες δοκιμές να βρίσκουν χειρότερα αποτελέσματα.



Σχήμα 4.5: Μέσος όρος θέσης που βρέθηκε η καλύτερη λύση

Στο Σχήμα 4.6 βλέπουμε κατά μέσο όρο του κάθε αλγορίθμου και λογισμικού το χρόνο εκτέλεσης της βελτιστοποίησης. Παρατηρούμε ότι ο αλγόριθμος πλέγματος παρόλο που μπορεί να μην έβρισκε πάντα καλύτερο αποτέλεσμα από τους υπόλοιπους αλγορίθμους βλέπουμε ότι ο μέσος όρος της εκτέλεσης του είναι σχεδόν ο μισός. Το ίδιο ισχύει και για τα συστήματα λογισμικού Nomad και Dakota που ήταν αρκετά αναμενόμενο καθώς όπως αναφέρεται αρκετές φορές παραπάνω, επιδιώκουν πάντα η επόμενη δοκιμή να είναι καλύτερη δηλαδή λιγότερος χρόνος στη λύση του προβλήματος και έτσι λιγότερος χρόνος συνολικά.



Σχήμα 4.6: Μέσος όρος χρόνου εκτέλεσης αλγορίθμων

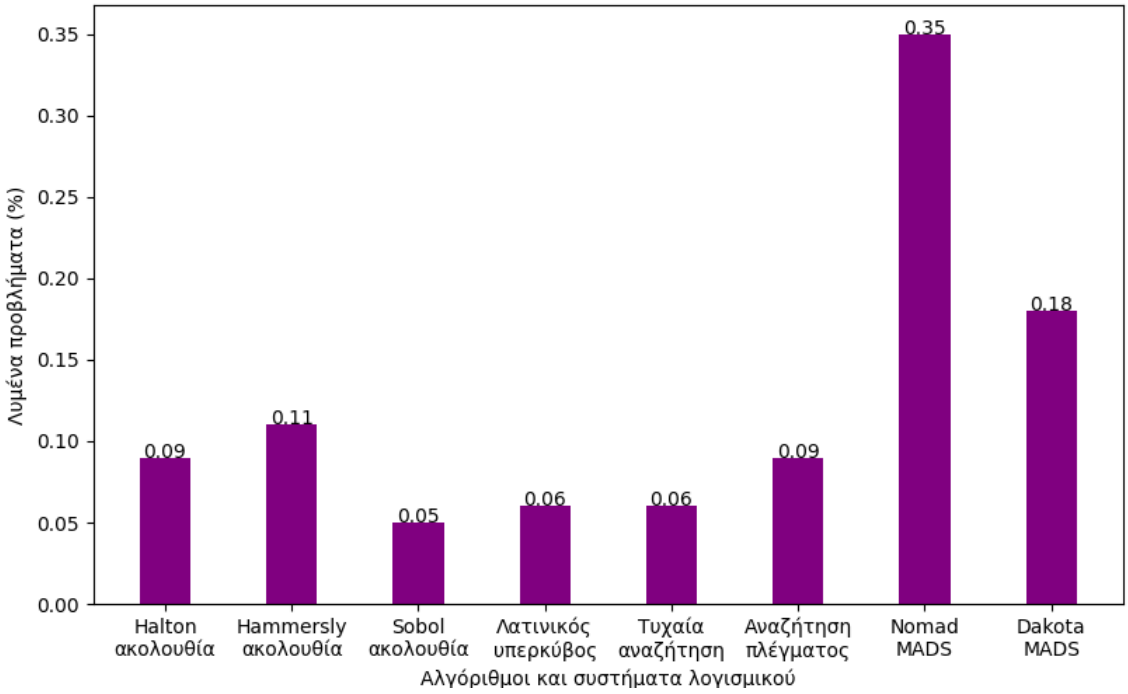
Αυτό που μπορούμε να βγάλουμε ως συμπέρασμα για την υπολογιστική μελέτη είναι το εξής: Τα συστήματα λογισμικού Nomad και Dakota έβρισκαν κατά μέσο όρο τις καλύτερες υπερπαραμέτρους και απαιτούσαν και λιγότερο χρόνο για την ολοκλήρωση των δοκιμών. Ο αλγόριθμος βασισμένος στο πλέγμα αναζήτησης είχε την καλύτερη απόδοση σε σχέση με τους άλλους αλγορίθμους, από ότι φαίνεται ο τρόπος υλοποίησης του ευνόησε τη συμπεριφορά του όπως αναφέρθηκε και στο κεφάλαιο 3.

Στον Πίνακα 4.4 παρουσιάζονται οι καλύτεροι χρόνοι που βρήκε ο κάθε αλγόριθμος και λογισμικό για κάθε πρόβλημα καθώς και η λύση με τις προεπιλεγμένες υπερπαραμέτρους. Παρατηρούμε ότι στην πλειοψηφία των αλγορίθμων κατά τη διαδικασία βελτιστοποίησης των προβλημάτων, βρέθηκαν καλύτερες λύσεις σε σχέση με τις λύσεις από τις προεπιλεγμένες τιμές υπερπαραμέτρων. Σε αρκετές περιπτώσεις βλέπουμε αρκετή βελτίωση στη λύση δηλαδή μείωση στο χρόνο εκτέλεσης του προβλήματος, ενώ σε άλλες περιπτώσεις έχουμε μικρή έως και ασήμαντη βελτίωση. Έχουμε και κάποιες εξαιρέσεις όπου μερικοί αλγόριθμοι δεν κατάφεραν να βελτιστοποιήσουν το πρόβλημα. Στα προβλήματα supportcase7, seymour1, swath3, neos-3988577-wolgan, blp-ic98 κάποιοι αλγόριθμοι δεν κατάφεραν να βελτιστοποιήσουν το χρόνο τους. Αν τα δούμε σε ποσοστά με βάση όλα τα προβλήματα έχουμε

τα εξής:

- Το ποσοστό των προβλημάτων που απέτυχε να βελτιστοποιήσει ο αλγόριθμος βασισμένος στην ακολουθία Halton είναι 1.54%
- Το ποσοστό των προβλημάτων που απέτυχε να βελτιστοποιήσει ο αλγόριθμος βασισμένος στην ακολουθία Sobol είναι 6.15%
- Το ποσοστό των προβλημάτων που απέτυχε να βελτιστοποιήσει ο αλγόριθμος βασισμένος στο λατινικό υπερκύβο είναι 3.07%
- Το ποσοστό των προβλημάτων που απέτυχε να βελτιστοποιήσει ο αλγόριθμος βασισμένος στην τυχαία αναζήτηση είναι 4.6%
- Όλοι οι υπόλοιποι αλγόριθμοι και τα συστήματα λογισμικού είχαν ποσοστό επιτυχίας βελτιστοποίησης 100%

Στο Σχήμα 4.7 παρουσιάζονται τα ποσοστά του συνόλου των προβλημάτων που ο κάθε αλγόριθμος ή λογισμικό κατάφερε να έχει την καλύτερη λύση από όλους τους υπόλοιπους.



Σχήμα 4.7: Ποσοστό προβλημάτων με καλύτερο χρόνο ανά αλγόριθμο

Πίνακας 4.4: Αποτελέσματα βελτιστοποίησης

Προβλήματα	**Halton**	**Hammerly**	**Sobol**	**Grid**	**Random**	**LHS**	**Dakota MADS**	**Nomad MADS**	**Default**
30n20b8	1.29	1.31	1.22	1.49	1.23	1.14	1.18	1.15	2.98
CMS740_4	1.28	1.86	2.34	1.25	2.65	2.44	0.83	0.82	3.1
air04	3.94	3.91	4.25	5.28	4.98	4.15	4.01	4.48	8.61
air05	3.81	4.48	3.32	4.56	3.57	3.36	3.68	3.40	9.01
app1-1	0.5	0.46	0.47	0.64	0.36	0.57	0.47	0.39	1.02
blp-ic98	31.25	26.88	33.91	27.38	30.72	28.4	25.4	21.08	33.65
cbs-cta	0.39	0.41	0.39	0.37	0.36	0.41	0.34	0.37	1.32
comp07-2idx	27.52	18.49	25.88	26.3	17.97	26.35	16.95	13.41	46.16
dano3_3	9.42	9.71	9.71	7.89	10.93	10.63	7.38	8.19	15.98
drayage-25-23	0.91	1.16	1.22	0.86	1.14	1.15	1.17	1.40	3.25
eil33-2	2.64	2.79	3.05	2.8	2.59	2.46	2.16	1.65	3.61
ex10	17.21	17.81	17.64	17.71	17.89	18.01	17.7	17.1	24.51
ex9	4.4	4.44	4.35	4.3	4.35	4.31	4.46	4.37	5.2
fast0507	27.5	33.71	23.69	31.7	31.03	30.52	23.2	30.07	41.69
fastxgemm-n2r6s0t2	14.13	15.29	22.33	15.46	14.02	14.12	13	12.37	24.95
fiball	1.13	1.89	1.82	2.21	1.6	1.94	2.11	0.85	3.34
hypothyroid-k1	11.42	11.75	11.27	15.54	15.86	12.99	15.58	13.03	15.89
irp	0.83	0.7	0.86	0.91	0.84	0.84	0.73	0.76	1.64

istanbul-no-cutoff	29.84	24.17	31.42	25.49	22.85	23.02	19.82	25.49	32.53
mcsched	30.13	21.38	26.49	16.31	29.12	24.78	20.56	15.84	45.96
mzzv11	4.17	3.9	4.12	3.66	3.63	4.19	3.5	4.03	7.38
mzzv42z	2.02	2.06	1.92	2.21	2.03	1.82	1.99	1.97	2.31
n2seq36q	1.28	1.34	1.23	0.85	1.07	2.38	1.15	1.15	3.7
neos-1122047	4.05	3.97	4.0	3.83	3.94	4.02	3.91	4	5.97
neos-1171448	0.47	0.48	0.48	0.46	0.48	0.45	0.45	0.44	2.07
neos-1171737	1.66	2.28	2.77	3.07	1.97	2.72	2.82	4.03	22.26
neos-1445765	15.21	16.7	18.1	9.08	18.31	16.61	9.34	7.99	18.75
neos-1582420	1.16	2.77	2.89	1.08	2.78	2.95	2.04	2.38	4.55
neos-2746589-doon	27.74	18.79	27.48	22.67	26.82	27.26	23.44	22.49	28.1
neos-2987310-joes	1.47	1.43	1.44	1.5	1.49	1.56	1.46	1.52	2.72
neos-3004026-krka	10.81	10.25	10.96	9.73	2.78	3.82	10.9	6.90	12.42
neos-3988577-wolgan	51.88	47.57	41.23	36.07	54.78	51.24	51.1	30.89	52.36
neos-4413714-turia	15.89	16.14	22.32	18.42	24.7	19.04	20.27	16.49	38.01
neos-4722843-widden	37.65	43.57	36.03	59.98	42.84	47.06	36.36	34.15	72.38
neos-4738912-atrato	7.42	7.9	7.76	8.24	7.84	6.48	6.55	7.32	41.28
neos-5107597-kakapo	25.8	22.92	28.31	33.93	22.28	33.67	17.3	12.17	59.62
neos-662469	17.23	5.88	6.67	12.17	8.85	9.61	5.03	7.53	49.91
neos-860300	4.8	5.68	4.94	5.58	5.88	5.33	4.88	5.3	13.53

neos-933966	1.9	1.25	1.79	0.94	1.86	1.84	0.91	0.86	3.52
neos-950242	5.39	5.51	6.01	8.62	5.99	5.84	7.05	6.08	12.88
neos-957323	2.56	2.54	2.66	2.62	2.54	2.55	2.6	2.5	3.39
neos-960392	1.61	1.13	1.8	1.53	1.71	1.82	1.6	1.54	3.04
net12	22.31	21.88	17.26	30.05	23.23	21.46	12.3	22.08	37.7
netdiversion	20.26	43.76	30.62	44.41	30.2	28.61	44.92	31.47	64.39
ns1208400	2.29	2.14	2.44	2.99	1.89	2.24	2.09	3.05	9.82
ns1830653	33.7	35.51	35.05	29.79	32.3	34.84	33.83	31.94	88.18
ns1952667	10.16	10.14	10.79	9.34	2.75	9.95	12.18	2.96	26.73
nu25-pr12	0.67	0.7	0.95	0.9	0.8	0.75	0.58	0.71	1.66
nurseched-sprint02	2.31	2.84	2.6	2.38	2.77	2.11	2	2.64	4.79
nw04	1.25	1.31	1.2	1.32	1.31	1.25	1.29	1.25	2.61
pg5_34	12.75	11.97	10.6	11.29	8.53	11.84	6.25	8.95	25.72
piperout-27	1.17	1.19	1.23	1.57	1.3	1.27	1.32	1.24	3.24
piperout-8	1.05	0.98	1.08	1.19	1.14	1.09	1.09	1.09	2.85
qap10	10.86	8.58	10.65	9.63	8.29	8.85	8.94	7.58	28.48
rmatr100-p10	10.1	8.51	10.2	7.67	9.57	7.7	5.56	5.38	11.6
rocl-4-11	28.26	35.05	31.47	33.4	31.54	34.73	31.12	23.36	76.57
seymour1	25.82	22.35	37.76	27.27	44.03	48.0	26.42	24.67	31.81
supportcase33	18.45	20.91	19.44	19.0	20.68	18.27	17.84	16.71	26.13

supportcase7	5.57	5.41	5.78	5.14	5.75	5.81	5.19	4.82	5.52
swath1	2.05	2.59	2.01	1.73	1.77	2.16	1.33	1.42	3.59
swath3	7.55	9.47	12.01	8.16	8.87	7.69	9.18	7.32	10.76
traininstance6	1.05	1.37	2.55	1.83	2.03	1.06	1.40	0.95	7.76
triptim1	21.4	18.77	23.67	18.26	19.48	16.76	18.95	18.18	25.23
uccase12	4.45	4.52	4.77	4.23	4.62	4.46	4.15	4.46	10.27
unitcal_7	20.94	19.46	20.46	18.56	19.12	20.02	18.52	16.47	21.54

Κεφάλαιο 5

Συμπεράσματα

Στα πλαίσια της διπλωματικής εργασίας ασχοληθήκαμε με την αυτοματοποίηση της εύρεσης βέλτιστων υπερπαραμέτρων και το πρόβλημα αυτό το χειριστήκαμε ως ένα πρόβλημα βελτιστοποίησης. Περιγράφηκε η έννοια των υπερπαραμέτρων που είναι πολύ γνωστή ειδικά στη μηχανική μάθηση και τονίστηκε πόσο σημαντική είναι η επιλογή των υπερπαραμέτρων.

Στη συνέχεια, περιγράφηκαν μέθοδοι που έχουν προταθεί στη βιβλιογραφία και χρησιμοποιούνται για την επίτευξη της αυτοματοποιημένης εύρεσης βέλτιστων υπερπαραμέτρων. Πιο συγκεκριμένα, περιγράφηκαν δειγματοληπτικοί αλγόριθμοι, όπως ο αλγόριθμος αναζήτησης πλέγματος και τυχαίας αναζήτησης, Μπεϋζιανή βελτιστοποίηση, γενετικοί αλγόριθμοι και αλγόριθμοι βελτιστοποίησης χωρίς παράγωγους. Εκτός από τη βιβλιογραφική ανάλυση, στη διπλωματική εργασία αναπτύχθηκαν και αναλύθηκαν έξι αλγόριθμοι και χρησιμοποιήθηκαν δύο συστήματα λογισμικού για την αυτοματοποιημένη εύρεση βέλτιστων υπερπαραμέτρων.

Αρχικά, παρουσιάστηκαν οι αλγόριθμοι αναζήτησης πλέγματος και τυχαίας αναζήτησης που είναι από τις πιο διαδεδομένους μεθόδους αυτοματοποιημένης εύρεσης υπερπαραμέτρων. Στη συνέχεια, παρουσιάστηκαν τρεις αλγόριθμοι που ανήκουν στην κατηγορία ακολουθιών χαμηλής απόκλισης, συγκεκριμένα οι ακολουθίες Halton, Hammersly και Sobol. Μετέπειτα, παρουσιάστηκε ο αλγόριθμος λατινικός υπερκύβος και τέλος, τα δύο συστήματα λογισμικού Nomad και Dakota που χρησιμοποιούνται για την εύρεση βέλτιστων υπερπαραμέτων με τη χρήση του αλγορίθμου MADS. Ο στόχος ήταν η εύρεση βέλτιστων υπερπαραμέτρων που θα μείωναν το χρόνο εκτέλεσης προβλημάτων γραμμικού προγραμματισμού με τη χρήση του λύτη Gurobi.

Από την ανάλυση που προέκυψε μπορούμε να συμπεράνουμε ότι σχεδόν σε όλα τα μαθηματικά προβλήματα όλοι οι μέθοδοι που χρησιμοποιήθηκαν κατάφεραν να βρουν υπερπαραμέτρους που μείωναν το χρόνο επίλυσης. Επιπλέον, τα συστήματα λογισμικού Nomad και Dakota είχαν τα καλύτερα αποτελέσματα, δηλαδή τη μεγαλύτερη μείωση στο χρόνο στην επίλυση των προβλημάτων, ενώ οι αλγόριθμοι που υλοποιήθηκαν είχαν πολύ παρόμοια αποτελέσματα.

Υπάρχει αρκετός χώρος επέκτασης της μελέτης της συγκεκριμένης εργασίας. Ως πρώτο βήμα θα μπορούσαν να προστεθούν και να συγκριθούν περισσότεροι αλγόριθμοι και συστήματα λογισμικού σε μεγαλύτερο σύνολο προβλημάτων βελτιστοποίησης. Με περισσότερες δοκιμές και δεδομένα, δε θα επιβεβαίωνε μόνο τα αποτελέσματα στη μελέτη, αλλά αυτό θα μας οδηγούσε σε πιο αξιόπιστες αξιολογήσεις και σε πιο ευρύτερα συμπεράσματα. Επίσης, μια άλλη κατεύθυνση για μελλοντική μελέτη θα ήταν η επιλογή μεγαλύτερου συνόλου υπερπαραμέτρων ή/και διαφορετικών. Τέλος, θα μπορούσαν να χρησιμοποιηθούν και άλλοι λύτες γραμμικού προγραμματισμού και να γίνει σύγκριση μεταξύ τους.

Βιβλιογραφία

[1] N. Gould, "An introduction to algorithms for continuous optimization," 2006.

[2] D. Gao and H. Sherali, *Advances in Applied Mathematics and Global Optimization: In Honor of Gilbert Strang*. Advances in Mechanics and Mathematics, Springer US, 2009.

[3] P. Schratz, J. Muenchow, E. Iturritxa, J. Richter, and A. Brenning, "Hyperparameter tuning and performance assessment of statistical and machine-learning algorithms using spatial data," *Ecological Modelling*, vol. 406, pp. 109–120, 2019.

[4] A. R. G. Enas Elgeldawi, Awny Sayed and A. M. Zaki, "Hyperparameter tuning for machine learning algorithms used for arabic sentiment analysis," *MDPI*, November 2021.

[5] D. Lakhmiri, S. L. Digabel, and C. Tribes, "Hypernomad: Hyperparameter optimization of deep neural networks using mesh adaptive direct search," *ACM Trans. Math. Softw.*, vol. 47, jun 2021.

[6] J. Wu, X.-Y. Chen, H. Zhang, L.-D. Xiong, H. Lei, and S.-H. Deng, "Hyperparameter optimization for machine learning models based on bayesian optimizationb," *Journal of Electronic Science and Technology*, vol. 17, no. 1, pp. 26–40, 2019.

[7] Y. Tillé, *Sampling Algorithms*, pp. 1273–1274. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[8] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in Neural Information Processing Systems* (F. Pereira, C. Burges, L. Bottou, and K. Weinberger, eds.), vol. 25, Curran Associates, Inc., 2012.

[9] T. Beckers, "An introduction to gaussian process models," 2021.

[10] "Acquisition functions." `https://tune.tidymodels.org/articles/acquisition_functions.html`.

[11] P. A. Vikhar, "Evolutionary algorithms: A critical review and its future prospects," in *2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC)*, pp. 261–265, 2016.

[12] L. Tani, D. Rand, C. Veelken, and M. Kadastik, "Evolutionary algorithms for hyperparameter optimization in machine learning for application in high energy physics," *The European Physical Journal C*, vol. 81, p. 170, Feb 2021.

[13] H. Niederreiter, *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics, 1992.

[14] B. Gandar, G. Loosli, and G. Deffuant, "Sample dispersion is better than sample discrepancy for classification," tech. rep., Oct. 2010.

[15] W. Kenton, "Monte carlo simulation." https://www.investopedia.com/terms/m/montecarlosimulation.asp, 2022.

[16] P.-A. H. Tien-Tsin Wong, Wai-Shing Luk, "Sampling with Hammersley and Halton Points."

[17] I. "Krykova, ""evaluating of path-dependent securities with low discrepancy methods"," "thesis", "Worcester Polytechnic Institute", "100 Institute Road, Worcester MA 01609-2280 USA", "January" "2004".

[18] P. Bratley and B. L. Fox, "Algorithm 659: Implementing sobol's quasirandom sequence generator," *ACM Trans. Math. Softw.*, vol. 14, p. 88–100, mar 1988.

[19] "Sobol (quasi random) sequence simplified." http://www.deltaquants.com/sobol-sequence-simplified.

[20] "Latin hypercube sampling: Simple definition." https://www.statisticshowto.com/latin-hypercube-sampling/.

[21] S. Le Digabel, "Nomad: Nonlinear optimization with the mads algorithm," *ACM Trans. Math. Softw.*, vol. 37, p. 44, 01 2011.

[22] C. Audet and J. Dennis, "Mesh adaptive direct search algorithms for constrained optimization," *SIAM Journal on Optimization*, vol. 17, pp. 188–217, 01 2006.