Ατομική Διπλωματική Εργασία

Υλοποίηση και Αποτίμηση Αλγορίθμου για Περιβαλλοντικά Ουδέτερη Κατανάλωση Ενέργειας σε Έξυπνα Σπίτια

Νικόλας Πολυκάρπου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Υλοποίηση και Αποτίμηση Αλγορίθμου για Περιβαλλοντικά Ουδέτερη Κατανάλωση Ενέργειας σε Έξυπνα Σπίτια

Νικόλας Πολυκάρπου

Επιβλέπων Καθηγητής
Δημήτρης Ζεϊναλιπούρ

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2022

Ευχαριστίες

Πρώτα απ’ όλα θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δρ. Δημήτρη Ζεϊναλιπούρ για την καθοδήγηση που μου παρείχε καθ’ όλη την διάρκεια της εκπόνησης της Ατομικής Διπλωματικής μου Εργασίας. Θα ήθελα επίσης να ευχαριστήσω τον κ. Σωτήρη Κωνσταντίνου, διδακτορικό φοιτητή του Πανεπιστημίου Κύπρου, για την καθοδήγηση, υποστήριξη και για την συμβολή που είχε στην παρούσα διπλωματική εργασία. Ακόμη θα ήθελα να ευχαριστήσω τον Δρ. Ανδρέα Κωνσταντινίδη, καθηγητή Πληροφορικής του Πανεπιστημίου Frederick για την πολύτιμη βοήθεια που μου παρείχε κατά την ανάπτυξη του αλγορίθμου.

Επιπρόσθετα θα ήθελα να ευχαριστήσω την οικογένεια μου για την στήριξη που μου παρείχε, οικονομικά και ψυχολογικά τα τελευταία χρόνια όπου φοιτούσα στο Πανεπιστήμιο Κύπρου. Χωρίς την στήριξη τους δεν θα είχα καταφέρει να ανταπεξέλθω, και να αξιοποιήσω στο μέγιστο δυνατό επίπεδο τις δυνατότητες μου ώστε να ολοκληρώσω το πτυχίο μου.

Τέλος θα ήθελα να ευχαριστήσω τον κάθε ένα ξεχωριστά από το διδακτικό προσωπικό του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου διότι μέσα από τα μαθήματα τα οποία έχω ολοκληρώσει έχω πάρει τα κατάλληλα εφόδια, αρχικά για να ολοκληρώσω το πρόγραμμα σπουδών μου, αλλά και για τους μελλοντικούς μου στόχους.

Περίληψη

Πέραν πάσης αμφιβολίας ένα από τα κύρια προβλήματα που έχει η ανθρωπότητα ως στόχο να λύσει είναι η υπερθέρμανση του πλανήτη. Η μεγαλύτερη κινητήρια δύναμη της θέρμανσης είναι η εκπομπή των αερίων του θερμοκηπίου, εκ των οποίων πέραν του 66% αποτελείται από διοξείδιο του άνθρακα (CO2). Για την μείωση των εκπομπών ρύπων διοξειδίου του άνθρακα γίνονται καθημερινά προσπάθειες από όλες τις χώρες του κόσμου για εύρεση τρόπων με στόχο την μείωση του. Κύρια πηγή των εκπομπών αυτών είναι η καύση ορυκτών καυσίμων για την παραγωγή ενέργειας.

Στην παρούσα διπλωματική εργασία, το πρόβλημα το οποίο επιχειρείται να λυθεί είναι η μείωση των εκπομπών ρύπων διοξειδίου του άνθρακα από την κατανάλωση ενέργειας που γίνεται εντός της οικίας. Η ανάπτυξη των έξυπνων συσκευών, μαζί με το Διαδίκτυο των Πραγμάτων τα τελευταία χρόνια μας δίνουν την δυνατότητα να διαχειριστούμε πιο έξυπνα τις συσκευές αυτές, με τέτοιο τρόπο που να εξυπηρετεί κάποιους στόχους, όπως για παράδειγμα την μείωση των εκπομπών ρύπων. Για την επίτευξη του στόχου αυτού, επιχειρούμε να εκμεταλλευτούμε όσον το δυνατόν περισσότερη ενέργεια η οποία παράγεται από ανανεώσιμες πηγές ενέργειας, μειώνοντας παράλληλα τις ανάγκες του σπιτιού για να εισάγει ενέργεια από το ενεργειακό δίκτυο, κάτι που θα μειώσει και την εκπομπή ρύπων διοξειδίου του άνθρακα.

Για την επίλυση του προβλήματος το οποίο θέσαμε, αναπτύξαμε δύο εκδόσεις αλγορίθμου, την έκδοση GreenCap και την έκδοση GreenCapComfort. Ο αλγόριθμος χρησιμοποιεί την λογική του γενετικού αλγορίθμου. Πιο συγκεκριμένα, η αρχικοποίηση του αλγόριθμου γίνεται με ένα αριθμό από τυχαίες λύσεις, και τρέχει για ένα αριθμό από γενιές. Σε κάθε γενιά γίνονται τα βασικά βήματα ενός γενετικού αλγορίθμου, με επιπλέον δύο ευρετικές συναρτήσεις, έχοντας ως στόχο κάθε γενιά να είναι καλύτερη από την προηγούμενη. Στον αλγόριθμο GreenCap επιχειρούμε την μείωση της εισαγόμενης ενέργειας που γίνεται από το ενεργειακό δίκτυο, άρα παράλληλα και την εκπομπή ρύπων διοξειδίου του άνθρακα ενώ στον αλγόριθμο GreenCapComfort ο χρήστης δίνει και κάποιους κανόνες άνεσης, και ο αλγόριθμος επιχειρεί την μείωση της εισαγόμενης

ενέργειας από το ενεργειακό δίκτυο, χωρίς όμως να παραβιάζει τους κανόνες άνεσης τους οποίους δίνει ο χρήστης.

Για την αξιολόγηση των 2 εκδόσεων του αλγορίθμου, υλοποιήσαμε ακόμα 3 περιπτώσεις αλγορίθμων. Ο πρώτος αλγόριθμος επιχειρεί κάποιες απλοποιήσεις στα δεδομένα, χωρίς όμως να επηρεάζει τα αποτελέσματα. Ο δεύτερος αλγόριθμος εκτελεί τυχαία ανακατανομή της κατανάλωσης των συσκευών κατά την διάρκεια της μέρας. Ο τρίτος αλγόριθμος είναι η εύρεση της βέλτιστης χρονοδρομολόγησης της κατανάλωσης των συσκευών.

Με το πέρας της αξιολόγησης που έγινε για τους αλγόριθμους που υλοποιήσαμε, καταφέραμε να αυξήσουμε την ιδιοκατανάλωση από 21% στα αρχικά μας δεδομένα, στο 50% στον αλγόριθμο GreenCapComfort και 54% στον αλγόριθμο GreenCap. Επίσης πετύχαμε να μειώσουμε την εισαγόμενη ενέργεια από το ενεργειακό δίκτυο για την συνολική κάλυψη των ενεργειακών αναγκών της οικίας από το 78% που ήταν στα αρχικά μας δεδομένα στο 46% στον αλγόριθμο GreenCap και στο 49% στον αλγόριθμο GreenCapComfort. Επιπλέον με ταλγόριθμο GreenCapComfort πετυχαίνουμε να έχουμε σημαντική μείωση στην εισαγόμενη ενέργεια άρα και στην εκπομπή ρύπων CO2, χωρίς να επηρεάζεται σε μεγάλο βαθμό το επίπεδο άνεσης του χρήστη, το οποίο παραμένει περίπου 90%.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Κεφάλαιο 1

Εισαγωγή

1.1 Κίνητρο

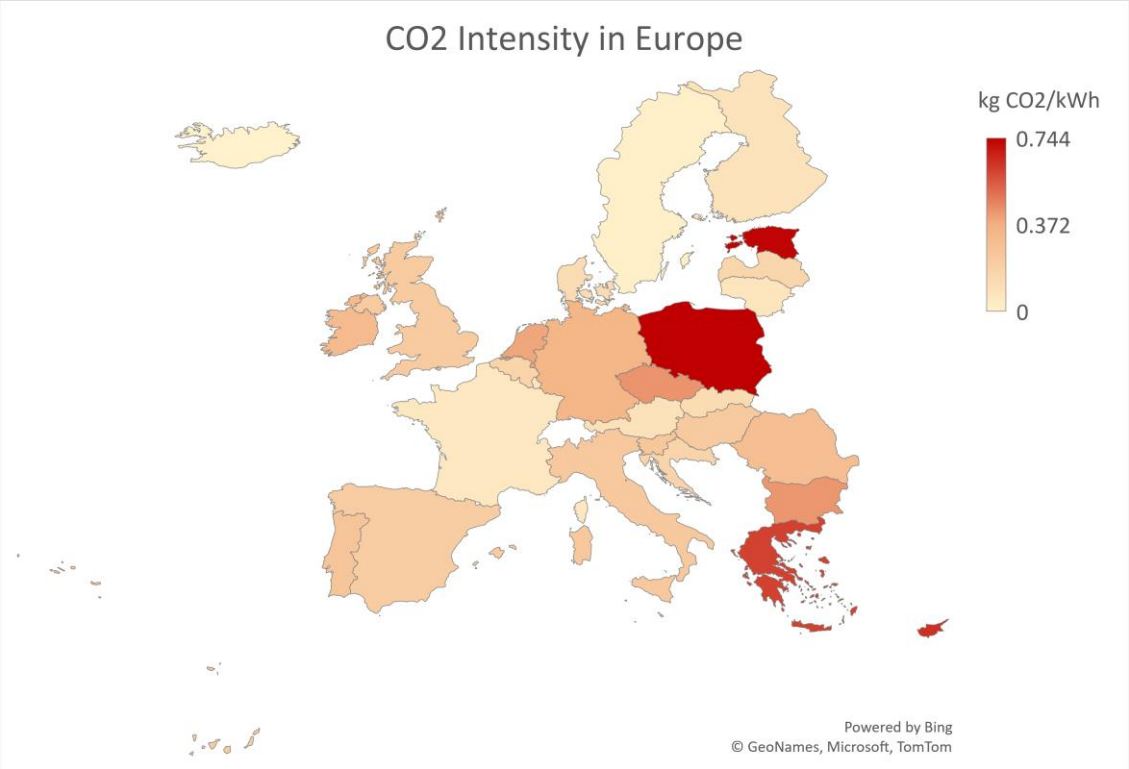
Αναμφίβολα ένα από τα κύρια ζητήματα που πρέπει να αντιμετωπίσει η ανθρωπότητα είναι η κλιματική αλλαγή και πιο συγκεκριμένα η υπερθέρμανση του πλανήτη. Ο κύριος μοχλός της κλιματικής αλλαγής είναι το φαινόμενο του θερμοκηπίου. Ορισμένα αέρια στην ατμόσφαιρα της γης λειτουργούν σαν γυαλί σε ένα θερμοκήπιο, παγιδεύοντας τη θερμότητα του ήλιου και εμποδίζοντας την να διαρρεύσει πίσω στο διάστημα, κάτι που είναι ικανό να προκαλέσει την υπερθέρμανση του πλανήτη. Πολλά από τα αέρια του θερμοκηπίου εμφανίζονται φυσικά, αλλά η ανθρώπινη δραστηριότητα αυξάνει τις συγκεντρώσεις ορισμένων από αυτά στην ατμόσφαιρα [1].

Η μεγαλύτερη κινητήρια δύναμη της υπερθέρμανσης είναι η εκπομπή αερίων, η οποία δημιουργά το φαινόμενο του θερμοκηπίου, εκ των οποίων περισσότερο από το 66% είναι το διοξείδιο του άνθρακα (CO2) [2]. Η καύση ορυκτών καυσίμων (άνθρακας, πετρέλαιο

και φυσικό αέριο) για κατανάλωση ενέργειας είναι η κύρια πηγή των εκπομπών διοξειδίου του άνθρακα (CO2).

Η οικιακή κατανάλωση είναι ένας από τους πιο σημαντικούς παράγοντες της κατανάλωσης ενέργειας και των εκπομπών ρύπων CO2. Η έκθεση της United State Energy Information Administration (EIA) για το 2021 ανέφερε ότι περίπου 1.5 εκατομμύρια kWh ηλεκτρικής ενέργειας πωλήθηκαν σε οικιακούς καταναλωτές το 2019, η αξία της οποίας ανέρχεται σε περίπου το 37.8% των συνολικών πωλήσεων ηλεκτρικής ενέργειας στους τέσσερις μεγάλους τομείς (οικιακή, εμπορική, βιομηχανική και μεταφορική). Επιπλέον, σύμφωνα με το [3], περίπου το 30% της παγκόσμιας συνολικής κατανάλωσης ενέργειας καταναλώνεται επί του παρόντος εντός σπιτιού.

Κάποιες χώρες έκαναν σημαντική βήματα ως προς την μείωση στην εκπομπή ρύπων διοξειδίου του άνθρακα από την κατανάλωση ενέργειας. Όπως μπορούμε να δούμε από το Σχήμα 1.1 όπου χώρες όπως η Φινλανδία ή η Σουηδία έχουν πολύ μικρή εκπομπή CO2 ανά kWh η οποία καταναλώνεται. Στόχος της Ευρωπαϊκής Ένωσης είναι η μείωση κατά 55% των εκπομπών αερίων του θερμοκηπίου έως το 2030 [5].



Σχήμα 1.1 Ένταση CO2 ανά kWh σε διάφορες χώρες της Ευρώπης [4]

Ακόμη ένα πρόβλημα το οποίο παρατηρείται, είναι τα πολύ ψηλά κόστη στην ηλεκτρική ενέργεια. Αυτό οφείλεται κυρίως στην κατανάλωση ενέργειας η οποία γίνεται κατά τις ώρες αιχμής, αφού σε αυτές τις ώρες το κόστος από την κατανάλωση ενέργειας είναι υψηλότερο απ' ότι στις υπόλοιπες ώρες. Η αύξηση τους κόστους στις ώρες αιχμής έχει ως στόχο την αποφόρτιση του ενεργειακού δικτύου κατά τις ώρες αιχμής, δίνοντας παράλληλα κίνητρο στους καταναλωτές έτσι ώστε να μειώσουν το κόστος από την κατανάλωση ενέργειας [6].

Επιπλέον, τα τελευταία χρόνια παρατηρείται πολύ μεγάλη ανάπτυξη στο διαδίκτυο των πραγμάτων (IoT), αφού η επιχειρηματική αγορά του IoT αυξήθηκε στα 157.9 δισεκατομμύρια δολάρια το 2021, ενώ αναμένεται μέχρι το 2027 να ξεπεράσει τα 525 δισεκατομμύρια [7]. Χρησιμοποιώντας έξυπνες συσκευές οι οποίες υποστηρίζονται από το διαδίκτυο των πραγμάτων, μας επιτρέπουν την μετατροπή των οικιών σε πιο έξυπνες. Με την χρησιμοποίηση του διαδικτύου των πραγμάτων στις οικίες μπορούμε να διαχειριζόμαστε άμεσα τις έξυπνες συσκευές μέσα στην οικία, κάτι που μας δίνει απεριόριστες δυνατότητες, όπως την παρακολούθηση και επεξεργασία της κατανάλωσης ενέργειας που γίνεται από τις έξυπνες συσκευές, την θερμοκρασία, την ποιότητα του αέρα, αν υπάρχει κίνηση εντός της οικίας κτλ. [8]

1.2 Παραδείγματα

Για την επίτευξη αυτού του στόχου της μείωσης των εκπομπών ρύπων διοξειδίου του άνθρακα υπάρχουν διάφορες λύσεις οι οποίες μπορούν να υιοθετηθούν. Αρχικά με την υιοθέτηση ανανεώσιμων πηγών ενέργειας για την παραγωγή της ενέργειας την οποία καταναλώνουμε. Υπάρχουν διάφοροι ανανεώσιμοι τρόποι που μπορεί να παραχθεί ενέργεια όπως με τον αέρα, με το νερό και με τον ήλιο.

Στα κτήρια και με την ενσωμάτωση ανανεώσιμων πηγών ενέργειας όπως τοποθέτηση φωτοβολταϊκών συστημάτων, θα μπορούσε να γίνεται ιδιοκατανάλωση της ηλιακής ενέργειας η οποία παράγεται. Επιπλέον, και στις μεταφορές υπάρχουν διάφορες λύσεις οποίες μπορούν να επιφέρουν την μείωση εκπομπών ρύπων CO2, όπως για παράδειγμα την χρησιμοποίηση καυσίμων με πιο βελτιωμένη απόδοση. Επίσης τα τελευταία χρόνια

υπάρχει τεράστια ανάπτυξη στα ηλεκτρικά αυτοκίνητα, τα οποία είναι «καθαρότερα» από τα βενζινοκίνητα ή τα πετρελαιοκίνητα αυτοκίνητα [9].

Επιπλέον με την χρήση του Demand-Side Management και την δυναμική τιμολόγηση της ηλεκτρικής ενέργειας δίνεται κίνητρο στους καταναλωτές να μεταφέρουν την κατανάλωση ενέργειας τους από τις ώρες αιχμής στις μη ώρες αιχμής έτσι ώστε να μειώσουν το κόστος της ηλεκτρικής ενέργειας το οποίο πρέπει να πληρώσουν. Η υιοθέτηση του διαδικτύου των πραγμάτων και με την ενσωμάτωση των έξυπνων συσκευών εντός της οικίας, η μεταφορά της κατανάλωσης ενέργειας από μία ώρα σε άλλη γίνεται ακόμα ευκολότερη.

1.3 Πρόβλημα

Το πρόβλημα το οποίο επιχειρεί να λύσει η παρούσα διπλωματική εργασία είναι με τον σχεδιασμό και την αξιολόγηση ενός αλγορίθμου για κλιματική ουδέτερη κατανάλωσης ενέργειας σε έξυπνα σπίτια, αρχικά την μείωση των εκπομπών ρύπων διοξειδίου του άνθρακα από την κατανάλωση ενέργειας σε μία οικία, και δεύτερο να εκμεταλλευτούμε στον μεγαλύτερο δυνατό βαθμό την δυναμική τιμολόγηση στην κατανάλωση ενέργειας μέσω του Demand Response έτσι ώστε το κόστος της ηλεκτρικής ενέργειας να είναι όσον το δυνατόν λιγότερο. Παράλληλα με την μείωση των εκπομπών ρύπων CO2 και τους κόστους της ενέργειας, επιχειρούμε μέσω κάποιων κανόνων άνεσης να διατηρήσουμε και το επίπεδο άνεσης του χρήστη όσον το δυνατόν ψηλότερα γίνεται.

1.4 Σχετική Εργασία

Πάνω στο πρόβλημα όπως περιεγράφηκε έγινε αρκετή έρευνα και σχεδιάστηκαν διάφορα συστήματα, όπου είχαν ως στόχο την μείωση των εκπομπών ρύπων διοξειδίου του άνθρακα ή την μείωση του κόστους της ηλεκτρικής ενέργειας ή και τα δύο μαζί, με κάποια εξ αυτών να λαμβάνουν υπόψιν και το επίπεδο άνεσης του χρήστη.

Για παράδειγμα, υπάρχουν έτοιμα συστήματα όπως το Sunny Home Manager από την SMA [10] ή το Bosch Energy Manager [11] τα οποία είναι διαθέσιμα αυτήν την στιγμή στην αγορά και μπορούν να εγκατασταθούν στην οικία σου, και με την έξυπνη διαχείριση των συσκευών της οικίας έχουν ως στόχο την εκμετάλλευση όσον το δυνατόν

περισσότερης ενέργειας η οποία παράγεται από το φωτοβολταϊκό σύστημα της οικίας. Επίσης το Google Nest Learning Thermostat [12] και το Ecobee Thermostat [13] είναι έξυπνοι θερμοστάτες οι οποίοι μπορούν να σου εξοικονομήσουν ενέργεια μέσω διαφόρων κανόνων οι οποίοι δίνονται από τους χρήστες έτσι ώστε να μην λειτουργά άσκοπα ο θερμοστάτης της οικίας. Επιπλέον με την εγκατάσταση έξυπνων πριζών στις συσκευές του σπιτιού ο ιδιοκτήτης μπορεί να παρακολουθεί την κατανάλωση ενέργειας η οποία γίνεται από τις συσκευές και να τις ανάβει/κλείνει με στόχο της μείωση της σπατάλης ενέργειας.

Για την παρακολούθηση και επεξεργασία της κατανάλωσης ενέργειας από τις συσκευές του σπιτιού αναπτύχθηκαν και διάφορα συστήματα αυτοματοποίησης, όπως το OpenHAB [16] και το Domoticz [17] όπου μέσω τον εφαρμογών αυτών των συστημάτων ο χρήστης μπορεί να παρακολουθεί την κατανάλωση ενέργειας που γίνεται από διάφορες συσκευές στην οικία και να αλλάξει την κατάσταση (ON/OFF) την οποία βρίσκονται οι συσκευές.

Στόχος αυτών των συστημάτων είναι η διαχείριση της ζήτησης ενέργειας, δηλαδή της αλλαγής στην κατανάλωση ηλεκτρικής ενέργειας. Αυτό επιτυγχάνεται με το load shifting, όπου μεταφέρεται η κατανάλωση ενέργειας από τις ώρες αιχμής στις μη ώρες αιχμής.

Αρκετή δουλειά έγινε και από την ερευνητική κοινότητα για την επίλυση του προβλήματος το οποίο θέσαμε προηγουμένως. Πιο συγκεκριμένα προτάθηκαν διάφοροι αλγόριθμοι για την αντιμετώπιση του προβλήματος το οποίο ορίσαμε προηγούμενος. Στο [26] προτάθηκε η χρήση Linear Programming για την μεταφορά του φόρτου της ηλεκτρικής ενέργειας από τις ώρες αιχμής στις μη ώρες αιχμής. Στο [29] και [30] οι συγγραφείς των δύο άρθρων προτείνουν την χρήση μοντέλων μηχανικής μάθησης και πιο συγκεκριμένα την χρήση νευρωνικών δικτύων (Neural Networks) για την διαχείριση ζήτησης ενέργειας. Επιπλέον στο [32] και [33] προτείνεται η χρησιμοποίηση του αλγόριθμου Mixed Integer Linear Programming για την βελτιστοποίηση της διαχείρισης ενέργειας στην οικία. Στο [35] προτείνεται η χρησιμοποίηση του αλγόριθμου Mixed Integer Non-Linear Programming και προτείνουν ένα σύστημα διαχείρισης ενέργειας στο σπίτι. Στο [37] και [38] προτείνεται η χρησιμοποίηση του προβλήματος του σακιδίου

(Knapsack Problem) για την ελαχιστοποίηση του κόστους της ηλεκτρικής ενέργειας σε μία οικία. Στο [41] και [42] προτείνεται η χρήση γενετικού αλγορίθμου για την επίλυση του προβλήματος ελαχιστοποίησης του κόστους ηλεκτρικής ενέργειας. Τέλος στο [44] προτάθηκε η χρησιμοποίηση δυναμικού προγραμματισμού για την υλοποίηση ενός πλαισίου διαχείρισης ενέργειας που θα έχει ως στόχο την μείωση του κόστους της ηλεκτρικής ενέργειας σε μία οικία.

1.5 IMCF

Η δουλειά που έγινε από τον προηγούμενο φοιτητή κατά την εκπόνηση της Ατομικής Διπλωματικής του Εργασίας είχε ως στόχο την επίλυση του προβλήματος της μείωσης του κόστους της ηλεκτρικής ενέργειας σε μία οικία κρατώντας το επίπεδο άνεσης του χρήστη ψηλά. Για την επίτευξη του στόχου αυτού θα έπρεπε να γίνει χρονοδρομολόγηση των συσκευών της οικίας σε διάφορες ώρες τις ημέρας. Ο αλγόριθμος ο οποίος σχεδιάστηκε παίρνει ως είσοδο κάποιους κανόνες λειτουργίας που δίνονται από τον χρήστη και κάποιο μακροχρόνιο στόχο. Ο αλγόριθμος αποτελείται από δύο υπορουτίνες, τον αλγόριθμο Amortization Plan (AP) το οποίο με βάση κάποιο σχέδιο απόσβεσης υπολογίζει κάποιον περιορισμό ως προς τον προϋπολογισμό της ενέργειας. Ακολούθως η επόμενη υπορουτίνα αποτελείται είτε από τον αλγόριθμο Energy Planner (EP) [45] είτε από τον αλγόριθμο Green Planner (GP) [46], όπου χρησιμοποιούν τους αλγόριθμους hill climbing και simulated annealing οι οποίοι είναι αλγόριθμοι εμπνευσμένοι από την αλγόριθμους τεχνητής νοημοσύνης, για την εύρεση της καλύτερης λύσης με βάση τους περιορισμούς που δίνονται. Η μόνη διαφορά στους δύο αλγόριθμους είναι ότι ο Green Plan λαμβάνει υπόψιν τις εκπομπές διοξειδίου του άνθρακα σε αντίθεση με τον Energy Plan ο οποίος λαμβάνει υπόψιν την κατανάλωση ενέργειας.

1.6 GreenCap

Ο αλγόριθμος GreenCap αναπτύχθηκε χρησιμοποιώντας την λογική του γενετικού αλγορίθμου. Πιο συγκεκριμένα, ο αλγόριθμος χρησιμοποιώντας την λογική του γενετικού αλγορίθμου επιτυγχάνουμε σε κάθε γενιά που τρέχει ο αλγόριθμος να έχουμε καλύτερες λύσεις, ως προς το σημείο αξιολόγησης το οποίο χρησιμοποιούμε. Σχεδιάσαμε δύο εκδόσεις του αλγόριθμου GreenCap, μία έκδοση που λαμβάνει υπόψιν το επίπεδο

άνεσης του χρήστη (GreenCapComfort), και μία έκδοση που δεν λαμβάνει υπόψιν την άνεση του χρήστη (GreenCap).

1.7 Πειράματα

Για την δοκιμή των δύο εκδόσεων του αλγορίθμου GreenCap υλοποιήθηκαν ακόμα τρεις αλγόριθμοι, ο αλγόριθμος Random όπου χρονοδρομολογεί τυχαία την κατανάλωση ενέργειας των συσκευών κατά την διάρκεια της ημέρας, με σεβασμό στην συνολική κατανάλωση η οποία γίνεται από την κάθε συσκευή ξεχωριστά κατά την διάρκεια της μέρας και στην μέγιστη κατανάλωση που μπορεί να κάνει μία συσκευή σε μία ώρα της μέρας. Ο δεύτερος αλγόριθμος ο οποίος υλοποιήθηκε είναι ο Brute Force ο οποίος χρησιμοποιεί DFS για την εύρεση της βέλτιστης χρονοδρομολόγησης της κατανάλωσης ενέργειας των συσκευών κατά την διάρκεια της μέρας. Επίσης υλοποιήθηκε ακόμα ένας αλγόριθμος, ο αλγόριθμος Standard, ο οποίος μετατρέπει τα αρχικά μας δεδομένα από μετρήσεις ανά λεπτό, σε μετρήσεις ανά ώρα για την απλούστευση των λειτουργιών των υπόλοιπων αλγορίθμων. Με πιο απλά λόγια, ο αλγόριθμος Standard επιστρέφει τα αρχικά δεδομένα.

Από τα πειράματα βρήκαμε ότι ο αλγόριθμος GreenCap επιτυγχάνει να μειώσει την εισαγόμενη ενέργεια της οικίας από το ενεργειακό δίκτυο, αυξάνει την ιδιοκατανάλωση που γίνεται από την ηλιακή ενέργεια από το φωτοβολταϊκό σύστημα και μειώνει την εκπομπή ρύπων διοξειδίου του άνθρακα. Πιο συγκεκριμένα, σε σχέση με τον αλγόριθμο Standard, όπου είναι τα αρχικά μας δεδομένα, η ιδιοκατανάλωση η οποία γίνεται είναι περίπου 21%, η εισαγόμενη ενέργεια είναι περίπου 78% ως προς την συνολική κατανάλωση που γίνεται. Ο αλγόριθμος GreenCap επιτυγχάνει να μειώσει την εισαγόμενη ενέργεια από το ενεργειακό δίκτυο, από το 78% που είναι αρχικά, σε περίπου 46%, και να αυξήσει την ιδιοκατανάλωση την ιδιοκατανάλωση από 21% σε 54%. Επίσης, ο αλγόριθμος GreenCapComfort, πετυχαίνει την μείωση της εισαγόμενης ενέργειας από 78% αρχικά, σε περίπου 49%, ενώ αυξάνει την ιδιοκατανάλωση από 21% σε περίπου 51%.

Σε αντίθεση με τον αλγόριθμο GreenCap ο αλγόριθμος GreenCapComfort επιτυγχάνει την μείωση της εισαγόμενης ενέργειας και παράλληλα την εκπομπή ρύπων διοξειδίου

του άνθρακα, όμως σε μικρότερο βαθμό από τον αλγόριθμο GreenCap. Ο αλγόριθμος GreenCapComfort όμως επιτυγχάνει να διατηρήσει το επίπεδο άνεσης του χρήστη, δηλαδή την υιοθέτηση των κανόνων άνεσης που δίνει ο χρήστης σε πολύ ψηλά επίπεδα, και πιο συγκεκριμένα πέραν του 90%, ενώ το επίπεδο άνεσης του χρήστη στον αλγόριθμο GreenCap είναι περίπου 30%.

1.8 Περίγραμμα

Η Ατομική Διπλωματική Εργασία αποτελείται από έξι κεφάλαια. Αναλυτικότερα τα έξι κεφάλαια είναι η «Εισαγωγή», η «Σχετική Εργασία», το «Ιστορικό του IMCF», το «GreenCap Framework», η «Πειραματική Μεθοδολογία και Αποτίμηση» και τα «Συμπεράσματα».

Στο παρόν κεφάλαιο (Κεφάλαιο 1 – Εισαγωγή) αναφέρθηκε και αναλύθηκε το κίνητρο το οποίο μας ώθησαν στην ανάπτυξη ενός αλγορίθμου για την κλιματική ουδέτερη κατανάλωσης ενέργειας σε έξυπνα σπίτια. Επίσης αναφέρθηκαν και οι σχετικές εργασίες οι οποίες έγιναν, καθώς επίσης έγινε και μία μικρή περίληψη του αλγορίθμου και των αποτελεσμάτων.

Στο δεύτερο κεφάλαιο (Κεφάλαιο 2 – Σχετική Εργασία) γίνεται μία περιγραφή των σχετικών εργασιών των οποίων έγιναν σε σχέση με το πρόβλημα το οποίο έχουμε να λύσουμε. Παρουσιάζονται υπάρχοντα προϊόντα που υπάρχουν αυτήν την στιγμή στην αγορά, και παρουσιάζονται διάφοροι αλγόριθμοι οι οποίοι σχεδιάστηκαν για παρόμοια προβλήματα.

Στο τρίτο κεφάλαιο (Κεφάλαιο 3 – Ιστορικό του IMCF) παρουσιάζεται η δουλειά που έγινε από τον προηγούμενο φοιτητή στα πλαίσια της δικής του Ατομικής Διπλωματικής Εργασίας. Το IMCF έχει ως στόχο την λύση παρόμοιου προβλήματος.

Στο τέταρτο κεφάλαιο (Κεφάλαιο 4 – GreenCap Framework) παρουσιάζει τον αλγόριθμο GreenCap. Αναλύονται πλήρως όλες οι διαδικασίες του αλγορίθμου, τα δεδομένα εισόδου του αλγορίθμου και τι επιστρέφει ως έξοδο.

Στο πέμπτο κεφάλαιο (Κεφάλαιο 5 – Πειραματική Μεθοδολογία και Αποτίμηση) αναλύεται πλήρως η μεθοδολογία που χρησιμοποιήθηκε για την εκτέλεση διαφόρων πειραμάτων. Πιο συγκεκριμένα, αναλύονται οι αλγόριθμοι που χρησιμοποιήθηκαν για την σύγκριση των αποτελεσμάτων και οι μετρικές που χρησιμοποιήθηκαν. Ακολούθως περιγράφεται η αξιολόγηση της απόδοσης του κάθε αλγόριθμου ξεχωριστά και έπειτα συγκριτικά μεταξύ όλων των αλγορίθμων.

Τέλος, στο έκτο κεφάλαιο (Κεφάλαιο 6 – Συμπεράσματα) αναφέρονται τα συμπεράσματα τα οποία βγήκαν μέσα από την δουλειά που έγινε. Επίσης γίνεται μία αποτίμηση της δουλειάς η οποία έγινε τους τελευταίους μήνες και τέλος αναφέρεται τι έμεινε ως μελλοντική εργασία.

Κεφάλαιο 2

Σχετική Εργασία

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τις σχετικές έρευνες που έγιναν στον χώρο της χρονοδρομολόγησης της λειτουργίας διάφορων έξυπνων οικιακών συσκευών με στόχο την μετατόπιση φορτίου από ώρες αιχμής σε ώρες μη αιχμής, αλλά και σε χρονοδρομολόγηση έξυπνων συσκευών με στόχο την εξοικονόμηση ενέργειας.

2.1 Έξυπνα συστήματα διαχείρισης ενέργειας

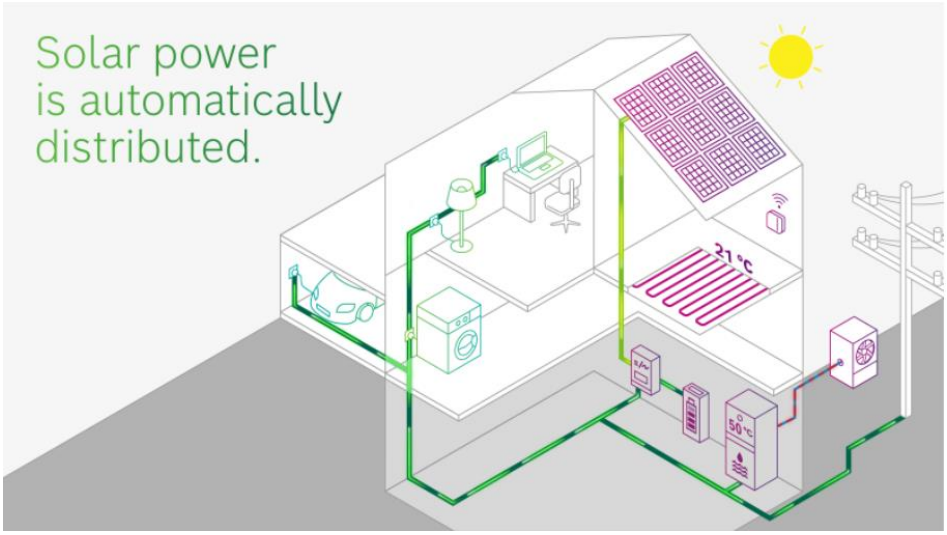
Τα τελευταία χρόνια με την υιοθέτηση του διαδικτύου των πραγμάτων (Internet of Things), και την εισαγωγή έξυπνων συσκευών σε όλες τις οικίες, η μεταφορά δεδομένων από και προς αυτές έγινε ευκολότερη και έτσι έδωσε την ευκαιρία στο να γίνει ευκολότερη η χρονοδρομολόγηση τους με στόχο την μείωση του κόστους από την κατανάλωσης ενέργειας. Αρκετές έρευνες οι οποίες έγιναν, καθώς και κάποιες λύσεις που υπάρχουν στην αγορά σήμερα αναλύονται παρακάτω.

2.1.1 Υπάρχοντα συστήματα στην αγορά

Ο Sunny Home Manager από την SMA [10] παρακολουθεί όλες τις ροές ενέργειας στο σπίτι και δημιουργά ένα έξυπνο σύστημα, το οποίο μπορεί να ελέγξει αυτοματοποιημένα τις συσκευές μέσα στην οικία, με στόχο την μέγιστη δυνατή εκμετάλλευση της ηλιακής ενέργειας η οποία παράγεται από το φωτοβολταϊκό σύστημα το οποίο υπάρχει εγκατεστημένο στο σπίτι. Αυτό γίνεται χρησιμοποιώντας την πρόγνωση του καιρού και την ανάλυση της κατανάλωσης ενέργειας που γίνεται μέσα στην οικεία, και έτσι δημιουργά μία ημερήσια απόδοση παραγωγής. Ο Sunny Home Manager αποτελείται από το φωτοβολταϊκό σύστημα, τον Sunny Home Manager ο οποίος φαίνεται στο Σχήμα 2.1 και τις έξυπνες συσκευές οι οποίες υπάρχουν στο σπίτι. Προαιρετικά μπορεί να χρησιμοποιηθεί και μπαταρία για αποθήκευση ηλιακής ενέργειας. Η επικοινωνία μεταξύ του manager και των συσκευών γίνεται με το πρωτόκολλο EEBus αν οι συσκευές το υποστηρίζουν, αλλιώς με το Simple Energy Management Protocol (SEMP).

Σχήμα 2.1 Sunny Home Manager 2.0 [10]

Παρόμοιο σύστημα είναι και το Bosch Energy Manager [11] το οποίο, με την χρήση ενός κεντρικού σημείου, διανέμεται η ηλιακή ενέργεια η οποία παράγεται από το φωτοβολταϊκό σύστημα, αρχικά στις έξυπνες συσκευές του σπιτιού, έτσι ώστε να καλυφτούν οι ανάγκες τους, ακολούθως στην αντλία θερμότητας της οικίας, και η περίσσεια ηλιακή ενέργεια αποθηκεύεται στην μπαταρία που υπάρχει στο σπίτι. Αξιοσημείωτο είναι ότι η εταιρία αναφέρει ότι μπορεί να καλυφτεί το 50% της ανάγκης του σπιτιού από την ηλιακή ενέργεια η οποία παράγεται, ενώ με την χρήση μπαταρίας μπορεί να καλυφτεί μέχρι και το 70% των αναγκών του σπιτιού(υποθέτοντας μία οικία τεσσάρων ατόμων με φωτοβολταϊκό σύστημα 6 kWp και αποθήκευση μπαταρίας 8 kWh.



Σχήμα 2.2 Bosch Energy Manager [11]

Το Google Nest Learning Thermostat [12], είναι έξυπνος θερμοστάτης ο οποίος μαθαίνει το πρόγραμμα σου και προγραμματίζεται με βάση αυτό, με στόχο την εξοικονόμηση ενέργειας. Πιο συγκεκριμένα, μπορείς να παρακολουθήσεις και να προγραμματίσεις την θέρμανση και την ψύξη εντός της οικίας καθώς και να παρακολουθεί πόση ενέργεια καταναλώθηκε από το σύστημα θέρμανσης και ψύξης. Επιπρόσθετα με την χρήση αισθητήρων, ανιχνεύει αν βρίσκεται κάποιος εντός της οικίας και ανάλογα προσαρμόζει την θερμοκρασία της οικίας, εξοικονομώντας με αυτό τον τρόπο ενέργεια, αποτρέποντας την σπατάλη ενέργειας.

Σχήμα 2.3 Google Nest Learning Thermostat [12]

Παρόμοιο σύστημα είναι και το Ecobee Thermostat [13], το οποίο είναι έξυπνος θερμοστάτης, όπου με την χρήση αισθητήρων μπορεί να ανιχνεύει αν υπάρχουν άτομα εντός της οικίας αλλά και αν υπάρχουν ανοιχτά παράθυρα έτσι ώστε να αποτρέπεται η άσκοπη λειτουργία του με αποτέλεσμα την μη σπατάλη ενέργειας. Ο έλεγχος του θερμοστάτη γίνεται μέσω της εφαρμογής. Η διαφορά με το Google Nest Learning Thermostat είναι ότι το πρόγραμμα λειτουργίας του θερμοστάτη καθορίζεται από τον χρήστη, σε αντίθεση με το Google Nest Learning Thermostat το οποίο παρέχει την δυνατότητα να δημιουργηθεί αυτόματα το πρόγραμμα λειτουργίας του θερμοστάτη.

Σχήμα 2.4 Ecobee Thermostat [13]

2.1.2 Έξυπνες οικιακές συσκευές

Οι έξυπνες οικιακές συσκευές είναι συνδεδεμένες με ένα κεντρικό σύστημα από το οποίο μπορούν να προγραμματιστούν ή να ελέγχεται η λειτουργία τους εξ αποστάσεως. Ο έλεγχος μπορεί να γίνει μέσω ενός συστήματος αυτοματοποίησης όπως αναλύεται παρακάτω ή μέσω της δικής του εφαρμογής όπου μας παρέχει η εταιρία παραγωγής. Οι περισσότερες έξυπνες συσκευές έρχονται με ενσωματωμένους αισθητήρες που ανιχνεύουν πράγματα και καταστάσεις όπως θερμοκρασία, επίπεδα φωτός, ή αν υπάρχει κίνηση δίπλα από αυτές κτλ. [14]

Δεν χρειάζεται να αγοράσουμε καινούργιες συσκευές σε περίπτωση που θέλουμε να μετατρέψουμε την οικία μας σε έξυπνη οικία, αρκεί μόνο να αγοράσουμε έξυπνες πρίζες οι οποίες μπορούν να τοποθετηθούν στις υπάρχοντες συσκευές μας και μέσω αυτών να παρακολουθούμε την κατανάλωση που γίνεται από τις οικιακές συσκευές.

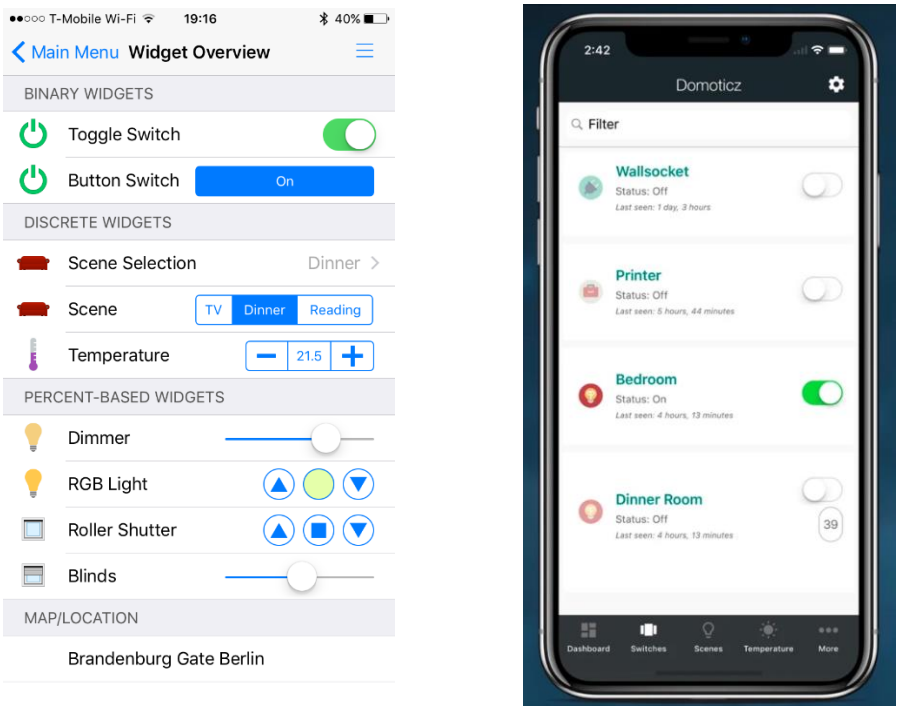
Σχήμα 2.5 Wemo Wi-Fi Smart Plug [15]

2.1.3 Συστήματα Αυτοματοποίησης

Αρκετά συστήματα αυτοματοποίησης έχουν αναπτυχθεί έτσι ώστε ο χρήστης να μπορεί να χειρίζεται τις συσκευές της οικίας του, όπου για παράδειγμα να μπορεί ανάβει ή/και να κλείνει συσκευές. Επίσης μέσω αυτών των συστημάτων ο χρήστης μπορεί να ορίζει κάποιους κανόνες και να λειτουργούν αυτοματοποιημένα οι συσκευές μίας οικίας. Για

παράδειγμα ένας κανόνας θα μπορούσε να ήταν να ανάψουν τα φώτα της κουζίνας μόλις νυχτώσει.

Το OpenHAB [16] είναι ένα σύστημα αυτοματοποίησης, το οποίο ο χρήστης μέσω της εφαρμογής μπορεί να δημιουργήσει κανόνες λειτουργίας συσκευών, οι οποίοι κανόνες θα ενεργοποιούνται με βάση κάποιο γεγονός (π.χ. αυξομείωση θερμοκρασίας, αλλαγή φωτεινότητας κτλ.) καθώς και να ανάψει ή/και να κλείσει συσκευές. Το openHAB υποστηρίζεται από ένα μεγάλο εύρος από συσκευές όπως το Amazon Alexa, Google Assistant, Daikin, Samsung κτλ.



Σχήμα 2.6 Κινητές Εφαρμογές OpenHAB[18] και Domoticz[19]

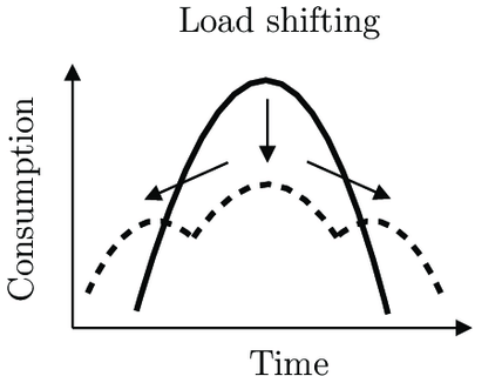
Ακόμα ένα παράδειγμα συστήματος αυτοματοποίησης είναι και το Domoticz [17], το οποίο σου επιτρέπει να παρακολουθείς και να χειριστείς διάφορες συσκευές (πχ. Φώτα, πρίζες κτλ.), διάφορους αισθητήρες(π.χ. Θερμοκρασία, Νερό κτλ.) και άλλα πολλά.

2.2 Διαχείριση από την πλευρά ζήτησης

Η διαχείριση της ζήτησης ενέργειας (Demand Side Management - DSM) είναι μία αλλαγή στην κατανάλωση ηλεκτρικής ενέργειας για την καλύτερη αντιστοίχιση της ζήτησης ενέργειας με την προσφορά [20]. Πιο συγκεκριμένα, με την χρησιμοποίηση δυναμικής τιμολόγησης στην τιμή της ηλεκτρικής ενέργειας δίνεται η ευκαιρία στους καταναλωτές να μειώσουν, ή να μετατοπίσουν την χρήση ηλεκτρικής ενέργειας από τις ώρες αιχμής στις μη ώρες αιχμής έτσι ώστε να μειώσουν τον λογαριασμό της ηλεκτρικής ενέργειας [21]. Υπάρχουν διάφορα τρόποι για την επίτευξη του DSM όπως το Energy efficiency, δηλαδή οι συσκευές που έχουν αρκετό κόστος στην ηλεκτρική ενέργεια να χρησιμοποιούνται πιο αποδοτικά (π.χ. να γεμίζουν τα πλυντήρια), ή με το με το Demand Response, δηλαδή την μείωση ή την μεταφορά του φόρτου ενέργειας από τις ώρες αιχμής σε μη ώρες αιχμής (Load Shifting).

2.3 Μετατόπιση Φορτίου (Load Shifting)

Με τον όρο load shifting εννοούμε την μετατόπιση κατανάλωσης ενέργειας από μία περίοδο σε μία άλλη. Η όλη ιδέα πίσω από τον όρο load shifting είναι με την μετατόπιση φορτίου από μια ώρα σε μία άλλη, τα πλεονεκτήματα από την εξοικονόμηση κόστους ενέργειας ή του Demand Response όπως αναλύθηκε προηγουμένως είναι περισσότερα από την απώλεια της παραγωγής ηλιακής ενέργειας εάν υπάρχει σε μία οικία. Σε αντίθεση με το Demand Response, το load shifting αντιμετωπίζει το αίνιγμα «πότε» και όχι το «πόσο» [22]. Ένα παράδειγμα για την μετατόπιση φορτίου από μία ώρα σε μία άλλη είναι η χρησιμοποίηση του θερμοσίφωνα κατά η ώρα 1μμ, αφού το μεσημέρι δεν υπάρχει πολύ μεγάλη ζήτηση ενέργειας, παρά την χρησιμοποίηση του θερμοσίφωνα η ώρα 7μμ που θεωρείται ώρα αιχμής.



Σχήμα 2.7 Load shifting [23]

2.4 Έρευνες, δημοσιεύσεις και αλγόριθμοι

Αρκετές έρευνες έγιναν στο πεδίο των συστημάτων χρονοδρομολόγησης έξυπνων συσκευών, στο load shifting και στο Demand-Side Management, όπου ως στόχο είχαν την μείωση του κόστους της ηλεκτρικής ενέργειας. Στις έρευνες αυτές χρησιμοποιήθηκαν διάφοροι αλγόριθμοι και παρακάτω αναφέρονται και αναλύονται κάποιοι από αυτούς.

2.4.1 Γραμμικός Προγραμματισμός (Linear Programming)

Ο γραμμικός προγραμματισμός είναι μία τεχνική βελτιστοποίησης για ένα σύστημα γραμμικών περιορισμών και μία γραμμική objective function. Μέσω της objective function ορίζεται η ποσότητα που πρέπει να βελτιστοποιηθεί και ο στόχος του γραμμικού προγραμματισμού είναι να βρει τις τιμές των μεταβλητών που μεγιστοποιούν ή ελαχιστοποιούν την objective function [24]. Τα γραμμικά προγράμματα μπορούν να εκφραστούν σε κανονική μορφή ως ακολούθως.

$$
\begin{array}{ll}
\text{Find a vector} & \mathbf{x} \\
\text{that maximizes} & \mathbf{c}^T\mathbf{x} \\
\text{subject to} & A\mathbf{x} \leq \mathbf{b} \\
\text{and} & \mathbf{x} \geq \mathbf{0}.
\end{array}
$$

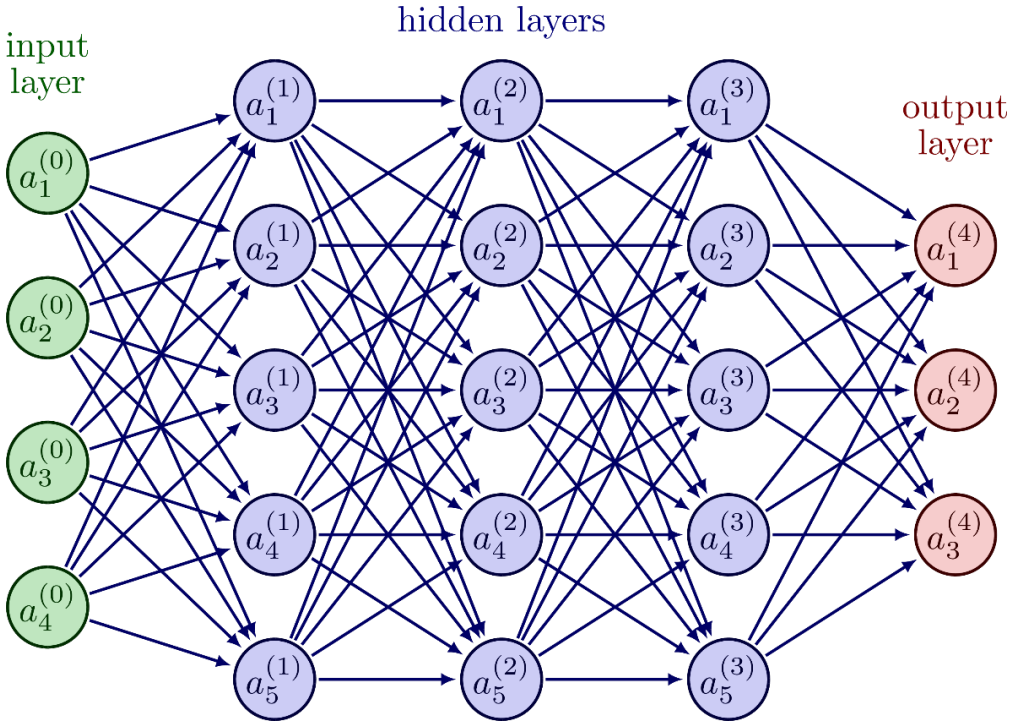
Σχήμα 2.8 Κανονική Μορφή Γραμμικού Προγραμματισμού[25]

Στο [26] οι συγγραφείς προτείνουν την ανάπτυξη ενός αλγορίθμου με στόχο την μεταφορά του φόρτου της ηλεκτρικής ενέργειας από ώρες αιχμής σε μη ώρες αιχμής, με σεβασμό στις προτιμήσεις των πελατών έτσι ώστε να μην επηρεαστεί το επίπεδο ικανοποίησης. Η μέθοδος την οποία υλοποιούν λειτουργεί σε πραγματικό χρόνο λαμβάνοντας πληροφορίες από τις ανανεώσιμες πηγές παραγωγής ενέργειας, καθώς και από την κατάσταση την οποία βρίσκονται οι συσκευές στην οικεία (ON/OFF) και με βάση αυτά βελτιστοποιείτε η κατανάλωση ενέργειας. Ο αλγόριθμος ο οποίος υλοποιήθηκε έγινε με βάση τον γραμμικό προγραμματισμό, δηλαδή ορίστηκαν κάποιοι κανόνες και αναλόγως ανάβουν ή σβήνουν οι συσκευές. Τα αποτελέσματα

επιβεβαιώνουν ότι μπορεί να επιτευχθεί βέλτιστη κατανομή ενέργειας για διάφορες καιρικές συνθήκες.

2.4.2 Μηχανική Μάθηση (Machine Learning)

Η μηχανική μάθηση είναι η μελέτη αλγορίθμων οι οποίοι μπορούν να βοηθήσουν στην αυτοματοποίηση ορισμένων διεργασιών. Οι αλγόριθμοι μηχανικής μάθησης χτίζουν μοντέλα τα οποία εκπαιδεύονται χρησιμοποιώντας ένα σύνολο δεδομένα, τα οποία ονομάζονται training data, με στόχο το μοντέλο το οποίο εκπαιδεύτηκε να μπορεί να προβλέπει καταστάσεις με βάση κάποια χαρακτηριστικά τα οποία αναγνωρίζει από τα δεδομένα [27]. Ένα παράδειγμα μοντέλου μηχανικής μάθησης είναι το ακόλουθο Artificial Neural Network (Multilayer Perceptron). Κάθε ακμή στο παρακάτω σχήμα έχει ένα βάρος. Με τον όρο εκπαίδευση, εννοούμε την εναλλαγή των βαρών μέχρι το άθροισμα των εισόδων επί τα βάρη, χρησιμοποιώντας μία activation function (πχ Sigmoid Function), να μας δίνει την αναμενόμενη έξοδο.



Σχήμα 2.9 Artificial Neural Network (Multilayer Perceptron) [28]

Στο [29] οι συγγραφείς προτείνουν επίσης την χρησιμοποίηση μηχανικής μάθησης για την δημιουργία ενός μοντέλου για την διαχείριση ζήτησης ενέργειας και του έξυπνου

συστήματος διαχείρισης ενέργειας. Πιο συγκεκριμένα, εστιάζονται σε 4 κατηγορίες φορτίων, οι οποίες περιλαμβάνουν κλιματιστικό, θερμοσίφωνα, πλυντήριο ρούχων και ψυγείο. Επέλεξαν να χρησιμοποιήσουν feed-forward Neural Network και τον αλγόριθμο Levenberg-Marquardt για την εκπαίδευση του μοντέλου. Το μοντέλο προβλέπει την βέλτιστη χρονοδρομολόγηση των προαναφερθέντων συσκευών. Η έρευνα τους έδειξε ότι με την χρησιμοποίηση του μοντέλου που πρότειναν επιτυγχάνουν την μείωση της κατανάλωσης ενέργειας χωρίς να επηρεάζεται ο τρόπος ζωής των πελατών.

Στο [30] οι συγγραφείς προτείνουν την χρησιμοποίηση μηχανικής μάθησης για την δημιουργία ενός μοντέλου για την διαχείριση ζήτησης ενέργειας και του έξυπνου συστήματος διαχείρισης ενέργειας. Τα φορτία από τις συσκευές χωρίζονται σε 3 κατηγορίες, τα σταθερά (π.χ. φώτα, υπολογιστές), τα ρυθμιζόμενα (π.χ. HVAC, θερμοσίφωνας) και τα αναβαλλόμενα φορτία (π.χ. πλυντήριο) και με βάση αυτά προτείνεται ένας αποσυνδεδεμένος μηχανισμός διαχείρισης ζήτησης ενέργειας για βέλτιστη διαχείριση. Τα δύο μοντέλα που χρησιμοποιούνται είναι το Neural Network Based Learning (Multi-layer Perceptron) και το Regression based Learning. Η έρευνα τους έδειξε ότι το μοντέλο που έχτισαν έδινε τα καλύτερα αποτελέσματα σε σύγκριση με τα άλλα μοντέλα τα οποία δοκίμασαν για να συγκρίνουν τα αποτελέσματα τους.

2.4.3 Μικτός Ακέραιος Γραμμικός Προγραμματισμός (Mixed Integer Linear Programming)

Οι αλγόριθμοι Mixed Integer Linear Programming είναι αλγόριθμος βελτιστοποίησης στον οποίο κάποιες μεταβλητές είναι ακέραιοι αριθμοί και κάποιες μη ακέραιοι εξού και το Mixed στο όνομα του. Το objective function του αλγόριθμου και οι περιορισμοί του είναι γραμμικοί. Για παράδειγμα το ακόλουθο πρόβλημα όπου θεωρείται η γενική μορφή ενός Integer Linear Programming προβλήματος [31].

$$
\begin{array}{lrcl}
\min & c^T x & & \\
\text{s.t.} & Ax & = & b \\
 & x & \geq & 0 \\
 & x & \in & Z^n
\end{array}
$$

Σχήμα 2.10 Γενική Μορφή Mixed Integer Linear Programming προβλήματος [31]

Στο [32] οι συγγραφείς προτείνουν την χρήση του Mixed Integer Linear Programming για την διαχείριση ζήτησης ενέργειας. Αναλυτικότερα, προτείνουν την χρήση ηλεκτρικών αυτοκινήτων ως χώρος αποθήκευσης ενέργειας και με την χρήση ενός έξυπνου διαχειριστή ενέργειας επιτυγχάνεται μείωση στην συνολική τιμή που χρειάζεται να πληρωθεί από μία οικεία για την κατανάλωση ενέργειας.

Στο [33] οι συγγραφείς προτείνουν όπως το πρόβλημα της βελτιστοποίησης για τη διαχείριση ενέργειας στο σπίτι να αναλυθεί σε ένα πρόβλημα βελτιστοποίησης δύο επιπέδων. Πιο συγκεκριμένα στο πρώτο επίπεδο να υπάρχει ένα καθολικό σύστημα διαχείρισης ενέργειας (GHEMS) όπου θα γίνεται χρονοδρομολόγηση της αποθήκευσης και της διακίνησης ενέργειας μεταξύ των σπιτιών, και στο δεύτερο επίπεδο ένα τοπικό σύστημα (LHEMS) που θα είναι τοποθετημένο σε κάθε οικεία όπου θα γίνεται χρονοδρομολόγηση των οικιακών συσκευών με βάση το πρόγραμμα και τις προτιμήσεις των χρηστών. Ο αλγόριθμος που χρησιμοποιείται είναι το Mixed Integer Linear Programming.

2.4.4 Μικτός Ακέραιος Μη Γραμμικός Προγραμματισμός (Mixed Integer Non-Linear Programming)

Οι αλγόριθμοι Mixed Integer Non Linear Programming είναι αλγόριθμος βελτιστοποίησης παρόμοιος με το Mixed Integer Linear Programming με την μόνη διαφορά ότι το objective function του αλγόριθμου ή/και οι περιορισμοί του είναι μη γραμμικοί. Για παράδειγμα το ακόλουθο πρόβλημα όπου θεωρείται η γενική μορφή ενός Mixed Integer Non Linear Programming προβλήματος, όπου το κάθε $c_i(x,y)$ είναι ένα mapping από το R^n στο R. Συνήθως οι συναρτήσεις f και c^i έχουν κάποιες ιδιότητες ομαλότητας, όπου για παράδειγμα μπορεί να είναι μία ή δύο φορές συνεχώς διαφοροποιήσιμες [34].

$$
\begin{array}{lllll}
\min & f(x,y) & & & \\
\text{s.t.} & c_i(x,y) & = & 0 & \forall i \in E \\
 & c_i(x,y) & \leq & 0 & \forall i \in I \\
 & x & \in & X & \\
 & y & \in & Y & \text{integer}
\end{array}
$$

Σχήμα 2.11 Γενική Μορφή Mixed Integer Non-Linear Programming προβλήματος [34]

Στο [35] οι συγγραφείς προτείνουν ένα σύστημα διαχείρισης ενέργειας στο σπίτι το οποίο ενσωματώνει διάφορες οικιακές συσκευές. Η όλη διαχείριση γίνεται με απόκριση σε δυναμική τιμολόγηση με στόχο την ελαχιστοποίηση του συνολικού κόστους της ηλεκτρικής ενέργειας. Χρησιμοποιώντας τον αλγόριθμο Mixed Integer Non-Linear Programming σε επαναλαμβανόμενη μορφή κατά την διάρκεια της μέρας έτσι ώστε να ακολουθείται η δυναμική τιμολόγηση του ηλεκτρικού ρεύματος επιτυγχάνεται η μείωση του κόστους της ηλεκτρικής ενέργειας, αλλά και διατηρείτε παράλληλα το επίπεδο άνεσης των χρηστών.

2.4.5 Πρόβλημα του Σακιδίου (Knapsack Problem)

Το πρόβλημα του σακιδίου (Knapsack problem) είναι πρόβλημα συνδυαστικής βελτιστοποίησης. Δεδομένου ενός συνόλου από αντικείμενα με το κάθε ένα να έχει ένα βάρος και μία τιμή, πρέπει να βρεθεί ο αριθμός των φορών που θα συμπεριληφθεί στη συλλογή έτσι ώστε το συνολικό βάρος να είναι μικρότερο ή ίσο με ένα δεδομένο όριο και η συνολική αξία να είναι όσο το δυνατό μεγαλύτερη [36]. Παρακάτω φαίνεται ένα παράδειγμα ενός Knapsack προβλήματος, όπου το W θεωρείται το όριο το οποίο δεν μπορεί να ξεπεραστεί.

$$\text{maximize} \sum_{i=1}^{n} v_i x_i$$

$$\text{subject to} \sum_{i=1}^{n} w_i x_i \leq W \text{ and } x_i \in \{0, 1, 2, \ldots, c\}.$$

Σχήμα 2.12 Γενική Μορφή Knapsack Problem [36]

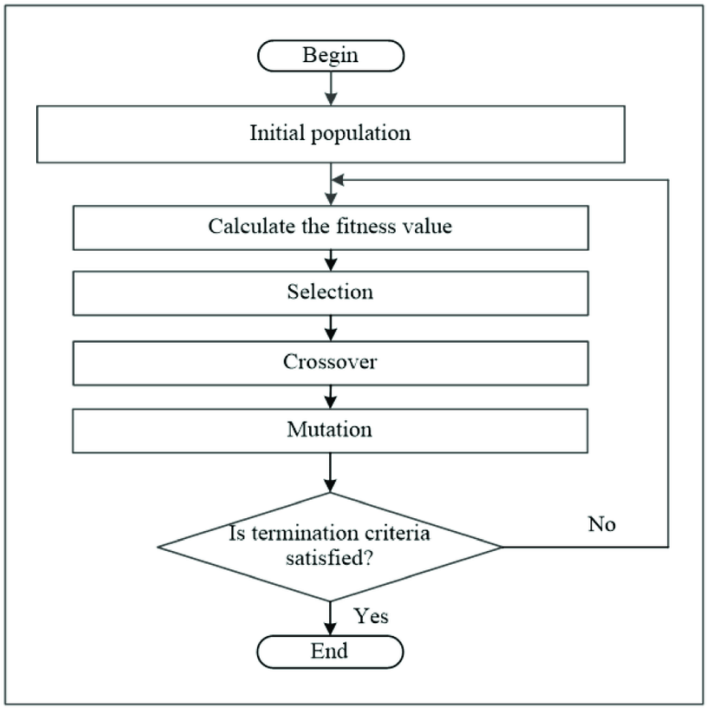
Στο [37] οι συγγραφείς προτείνουν την χρήση single Knapsack προβλήματος για την δημιουργία ενός μοντέλου ελαχιστοποίησης του συνολικού κόστους της ηλεκτρικής ενέργειας σε μία οικία με την χρήση συστήματος διαχείρισης ενέργειας το οποίο είναι ενσωματωμένο με ανανεώσιμες πηγές ενέργειας. Χρησιμοποιώντας τον αλγόριθμο Knapsack, υλοποιούν δύο παραδείγματα εκτέλεσης, ένα με ανανεώσιμες πηγές ενέργειας και μπαταρία αποθήκευσης ενέργειας και ένα χωρίς. Στόχος σε κάθε ένα από αυτά είναι να ελαχιστοποιήσει το κόστος της ηλεκτρικής ενέργειας, όμως παράλληλα να είναι ανοιχτές όσο περισσότερες συσκευές είναι δυνατόν. Κάθε συσκευή στο πρόβλημα

λαμβάνει υπόψιν τα Watts που κάνει κατανάλωση (weight) και το πόση ώρα είναι ανοιχτές (value). Επιτυγχάνει με την χρήση του αλγόριθμου Knapsack και στα δύο παραδείγματα να μειωθεί το κόστος, με μόνη διαφορά με την χρήση ανανεώσιμων πηγών ενέργειας και μπαταρίας να μην επηρεάζεται η άνεση του χρήστη.

Στο [38] οι συγγραφείς προτείνουν την χρήση Multiple Knapsack προβλήματος, και πιο συγκεκριμένα χρησιμοποιούν ένα Knapsack πρόβλημα για κάθε ώρα της μέρας έτσι ώστε να δρομολογηθούν διάφορες συσκευές στην οικία με στόχο την ελαχιστοποίηση του κόστους της ηλεκτρικής ενέργειας και της μεγιστοποίησης του επίπεδου άνεσης του χρήστη. Επίσης χρησιμοποιούν το Ant Colony Optimization (ACO) το οποίο είναι μία μεταευρετική τεχνική που επιτρέπει γρήγορο ρυθμό σύγκλισης για τον προγραμματισμό των συσκευών. Αναλυτικότερα το ACO μπορεί να βρει το μικρότερο μονοπάτι για κάποιον προορισμό. Σε κάθε Knapsack πρόβλημα χρησιμοποιείτε ως αντικείμενο η κάθε συσκευή, το βάρος της είναι η κατανάλωση ενέργειας που κάνει η συσκευή, και το value είναι το κόστος της κατανάλωσης ενέργειας που κάνει η συσκευή.

2.4.6 Γενετικός Αλγόριθμος (Genetic Algorithm)

Ο γενετικός αλγόριθμος είναι ένας ευρετικός αλγόριθμος αναζήτησης. Αντικατοπτρίζει τη διαδικασία φυσικής επιλογής όπου επιλέγονται τα πιο κατάλληλα άτομα για αναπαραγωγή προκειμένου να παραχθούν οι απόγονοι της επόμενης γενιάς. Ο αλγόριθμος αποτελείται από 5 στάδια όπως φαίνεται στο σχήμα παρακάτω [39]. Παραπάνω ανάλυση του αλγορίθμου θα γίνει στο κεφάλαιο 4 (Κεφάλαιο 4.2) .



Σχήμα 2.13 Γενετικός Αλγόριθμος [40]

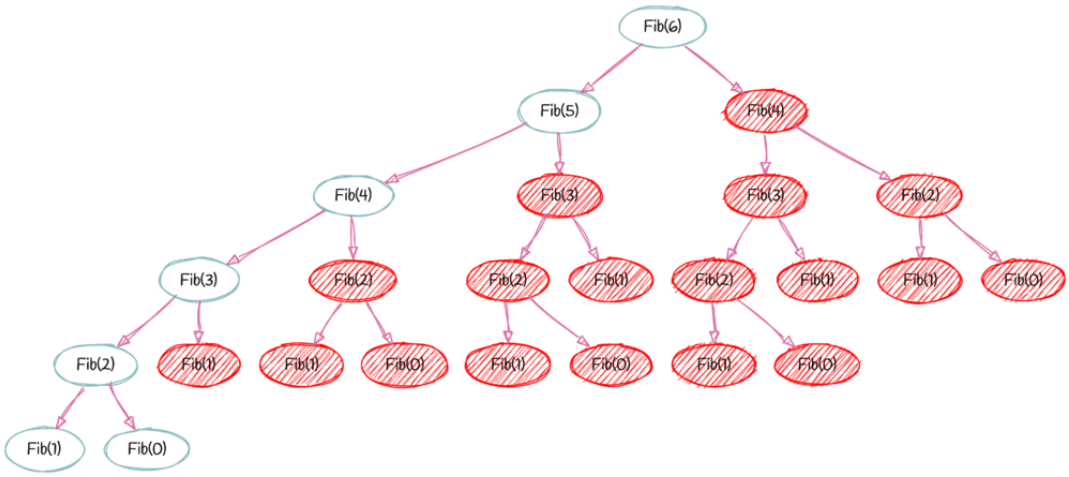
Στο [41] οι συγγραφείς προτείνουν ένα αλγόριθμο διαχείρισης ζήτησης με στόχο την ελαχιστοποίηση του κόστους της ηλεκτρικής ενέργειας με βάση την μεταφορά φόρτου από ώρες αιχμής σε μη ώρες αιχμής. Στην έρευνα τους λαμβάνουν υπόψιν δύο τύπους φορτίων, τα διακριτά και τα συνεχή. Το πρόβλημα το ορίζουν ως ένα 0-1 Knapsack πρόβλημα το οποίο αργότερα το επιλύουν με την χρήση γενετικού αλγορίθμου. Το fitness function που χρησιμοποιούν είναι το άθροισμα της κατανάλωση και το κόστος της ενέργειας ανά ώρα για όλες τις συσκευές της οικείας.

Στο [42] οι συγγραφεί προτείνουν έναν ευρετικό αλγόριθμο βελτιστοποίησης όπου έχει ως στόχο την ελαχιστοποίηση του κόστους της ηλεκτρικής ενέργειας, των κορυφών (peaks) στην ζήτηση, των εκπομπών ρύπων διοξειδίου του άνθρακα και την μεγιστοποίηση της άνεσης του χρήστη. Στην έρευνα αυτή λαμβάνονται υπόψιν και μπαταρία αποθήκευσης ενέργειας, φωτοβολταϊκό σύστημα και σύστημα διαχείρισης ενέργειας. Για την επίτευξη των προαναφερθέντων στόχων χρησιμοποιήθηκαν δύο παραδείγματα, τον γενετικό αλγόριθμο και ακολούθως τον HGPO αλγόριθμο όπως τον ονομάζουν, όπου είναι συνδυασμός των χαρακτηριστικών του γενετικού αλγόριθμου και του Particle Swarm Optimization. Με τα δύο παραδείγματα αλγορίθμων, επιτυγχάνουν την ελαχιστοποίηση του κόστους της ηλεκτρικής ενέργειας, των κορυφών στην ζήτηση,

των εκπομπών άνθρακα και την μεγιστοποίηση της άνεσης του χρήστη, όμως με καλύτερα αποτελέσματα να δίνει ο HGPO αλγόριθμος.

2.4.7 Δυναμικός Προγραμματισμός (Dynamic Programming)

Ο δυναμικός προγραμματισμός είναι αλγόριθμος για επίλυση προβλημάτων βελτιστοποίησης σπάζοντας τα σε απλούστερα υποπροβλήματα και χρησιμοποιώντας το γεγονός ότι η βέλτιστη λύση στο συνολικό πρόβλημα εξαρτάται από τη βέλτιστη λύση των υποπροβλημάτων [43].



Σχήμα 2.14 Δυναμικός Προγραμματισμός για το πρόβλημα του Fibonacci [52]

Στο [44] οι συγγραφείς προτείνουν την υλοποίηση ενός πλαισίου διαχείρισης ενέργειας το οποίο θα έχει ως στόχο την μείωση του κόστους της ηλεκτρικής ενέργειας. Η τεχνική η οποία χρησιμοποιούν έχει ως στόχο την μεταφορά φόρτου από τις ώρες αιχμής σε μη ώρες αιχμής, και ενσωματώνουν στο κτίριο και φωτοβολταϊκό σύστημα, ως μία λύση για εξοικονόμηση ενέργειας. Επίσης λαμβάνουν υπόψιν και τις προτιμήσεις των χρηστών σε ότι αφορά την χρονοδρομολόγηση των συσκευών του σπιτιού. Για την επίτευξη του στόχου που έχουν θέσει προτείνουν μία στρατηγική χρονοδρομολόγησης που έχει ως βάση τον δυναμικό προγραμματισμό. Επιτυγχάνουν με αυτήν την στρατηγική την μείωση κατά 82.3% του κόστους της ηλεκτρικής ενέργειας της οικίας.

2.4.8 Σύγκριση αλγορίθμων

Αναφέρθηκαν διάφοροι αλγόριθμοι οι οποίοι χρησιμοποιήθηκαν σε διάφορες έρευνες οι οποίες έγιναν, και δημοσιεύτηκαν. Κάθε ένας από αυτούς συνοδεύεται με κάποια πλεονεκτήματα αλλά και μειονεκτήματα. Όλοι οι αλγόριθμοι οι οποίοι αναφέρθηκαν, είχαν ως κύριο στόχο την μείωση του κόστους της ηλεκτρικής ενέργειας. Κάποιοι από αυτούς λαμβάνουν υπόψιν και το επίπεδο άνεσης του χρήστη όπως για παράδειγμα ο αλγόριθμος Knapsack, κάποιοι χρησιμοποιούν ανανεώσιμες πηγές ενέργειας όπως για παράδειγμα ο αλγόριθμος με βάση τον Δυναμικό προγραμματισμό, ενώ κάποιοι επιχειρούν να εκμεταλλευτούν το demand response όπως για παράδειγμα ο αλγόριθμος Mixed Integer Linear Programming. Το πιο ολοκληρωμένο άρθρο είναι το [42], και πιο συγκεκριμένα η χρήση γενετικού αλγορίθμου μαζί με επιπλέον ευρετικές συναρτήσεις όπου έχει ως στόχο με την χρησιμοποίηση ανανεώσιμων πηγών ενέργειας την επίλυση του προβλήματος έτσι όπως το ορίσαμε προηγουμένως, δηλαδή, την ελαχιστοποίηση του κόστους της ηλεκτρικής ενέργειας, των κορυφών στην ζήτηση, των εκπομπών ρύπων διοξειδίου του άνθρακα και παράλληλα την μεγιστοποίηση της άνεσης του χρήστη.

Κεφάλαιο 3

Ιστορικό του IMCF

Σε αυτό το κεφάλαιο θα αναλυθεί η δουλειά που έγινε στα πλαίσια της Ατομικής Διπλωματικής Εργασίας από τον προηγούμενο φοιτητή με θέμα την ανάπτυξη αλγορίθμου για εξοικονόμηση ενέργειας. Θα γίνει μία περίληψη του IoT Meta Control Firewall και ακολούθως θα αναλυθεί η αρχιτεκτονική του.

3.1 Περίληψη

Η προηγούμενη δουλειά που έγινε στα πλαίσια της Ατομικής Διπλωματικής Εργασίας του προηγούμενου φοιτητή είχε παρόμοιο θέμα με την παρούσα διπλωματική εργασία. Πιο συγκεκριμένα, το θέμα αφορούσε την ανάπτυξη και αξιολόγηση ενός αλγορίθμου για εξοικονόμηση ενέργειας σε έξυπνα σπίτια. Μέσα από την ερευνητική δουλειά η οποία έγινε και από άρθρα τα οποία δημοσιεύτηκαν, θα γίνει η περαιτέρω ανάλυση του IMCF στις επόμενες υποενότητες.

Με την ανάπτυξη των ανανεώσιμων πηγών ενέργειας αναδείχτηκε η σημαντικότητα της ιδιοκατανάλωσης της ενέργειας η οποίας παράγεται από τις ανανεώσιμες πηγές ενέργειας

μέσω των διάφορων έξυπνων συσκευών που υπάρχουν εντός της οικίας. Με αυτό τον τρόπο το ενεργειακό δίκτυο σταθεροποιείτε, ελαχιστοποιείτε η διασπορά ενέργειας και παράλληλα ελαχιστοποιούνται και οι εκπομπές ρύπων διοξειδίου του άνθρακα.

Στόχος του αλγόριθμου ο οποίος αναπτύχθηκε είναι να γεφυρωθεί το χάσμα μεταξύ της άνεσης του χρήστη, της κατανάλωσης ενέργειας και των εκπομπών διοξειδίου του άνθρακα (CO2). Για την επίτευξη του στόχου αυτού αναπτύχθηκε ένας καινοτόμος αλγόριθμος, ο Energy Planner (EP), ο οποίος είναι ένας αλγόριθμος εμπνευσμένος από την τεχνητή νοημοσύνη, που προγραμματίζει την κατανάλωση ενέργειας με μία ποικιλία στρατηγικών απόσβεσης (Amortization Plan). Ο αλγόριθμος του IMCF λαμβάνει υπόψιν κάποια Meta-rules (κανόνες) τα οποία δίνονται από τον χρήστη μέσω ενός συστήματος το οποίο κατασκευάστηκε έτσι ώστε ο χρήστης να εισάγει τους κανόνες.

3.2 Αρχιτεκτονική

Σε αυτή το υποκεφάλαιο θα αναλυθεί πλήρως η αρχιτεκτονική του IMCF, δηλαδή θα αναλυθούν όλα τα κομμάτια τα οποία συνθέτουν την αρχιτεκτονική του IMCF.

3.2.1 Αλγόριθμος

Ο αλγόριθμος αποτελείται από δύο υπορουτίνες, το αλγόριθμο Amortization Planner (AP) και είτε τον αλγόριθμο Energy Planner (EP) [45] είτε τον αλγόριθμο Green Planner (GP) [46]. Ο συνδυασμός αυτών των δύο, δηλαδή του AP μαζί με ένα εκ των EP ή GP αποτελεί τη διαδικασία διαχείρισης ενέργειας.

Στόχος του αλγόριθμου είναι να βρει ένα ενεργειακά αποδοτικό σχέδιο για την εκτέλεση ενός συνόλου από Meta-rules και ενός δοκιμαστικού ιστορικού κατανάλωσης ενέργειας (ECP - Energy Consumption Profile), και παράλληλα να ικανοποιεί πολλούς στόχους που υπόκεινται σε συγκεκριμένους μακροπρόθεσμους περιορισμούς είτε της συνολικής κατανάλωσης ενέργειας στον Energy Planner είτε της εκπομπής ρύπων CO2 στον Green Planner.

Οι μετρικές οι οποίες χρησιμοποιήθηκαν είναι το κόστος σφάλματος της άνεσης του χρήστη (Comfort Error - CE), η κατανάλωση ενέργειας (Ε) και η εκπομπή ρύπων διοξειδίου του άνθρακα (Γ).

3.2.2 Αλγόριθμος Amortization Planner

Ο αλγόριθμος Amortization Planner (σχέδιο απόσβεσης) είναι υπεύθυνος για τον υπολογισμό του μέγιστου ενεργειακού προϋπολογισμού και των μέγιστων εκπομπών CO2 μέσω ενός προεπιλεγμένου τύπου απόσβεσης. Υπάρχουν αρκετές στρατηγικές απόσβεσης οι οποίες μπορούν να χρησιμοποιηθούν. Οι στρατηγικές απόσβεσης οι οποίες χρησιμοποιήθηκαν στον αλγόριθμο είναι οι ακόλουθες.

Η πρώτη στρατηγική απόσβεσης που μπορεί να χρησιμοποιηθεί είναι η Linear Amortization Formula (LAF) όπου υπολογίζεται διαιρώντας την συνολική κατανάλωση ενέργειας με ένα διάστημα χρόνου, όπου αυτό το διάστημα μπορεί να είναι χρόνος, μέρα, ώρα κτλ. Αποτελεί την απλούστερη μορφή απόσβεσης επειδή χωρίζει το ποσό απόσβεσης ισόποσα σε ένα αριθμό από περιόδους.

Η δεύτερη δυνατή στρατηγική απόσβεσης που μπορεί να χρησιμοποιηθεί είναι η Balloon Linear Amortization Formula (BLAF). Σε αυτήν την περίπτωση ο χρήστης αποθηκεύει ένα ποσοστό ενέργειας από την συνολική ενέργεια σε για μια περίοδο, το καλούμενο μπαλόνι, το οποίο θα χρησιμοποιηθεί σε μία άλλη περίοδο όπου η κατανάλωση ενέργειας είναι ψηλότερη.

Η τρίτη δυνατή στρατηγική απόσβεσης που μπορεί να χρησιμοποιηθεί είναι η ECP-based Amortization Formula (EAF). Σε αυτή την περίπτωση, ένα σετ από βάρη υπολογίζονται χρησιμοποιώντας τον ECP vector. Τα βάρη στην συνέχεια χρησιμοποιούνται για να υπολογιστεί ο μέγιστος ενεργειακός προϋπολογισμός.

Ακολούθως αν θα χρησιμοποιηθεί ο αλγόριθμος Green Planner ανεξάρτητα από την επιλογή της στρατηγικής απόσβεσης υπολογίζεται και η μέγιστη εκπομπή ρύπων, πολλαπλασιάζοντας τον μέγιστο προϋπολογισμό ενέργειας με την ένταση ρύπων (kgCO2/kWh) της χώρας επιλογής του χρήστη.

3.2.3 Αλγόριθμος Energy Planner

Ο αλγόριθμο Energy Planner παίρνει ως είσοδος ένα πίνακα που περιέχει τα Meta-rules τα οποία δίνει ο χρήστης, τον μέγιστο αριθμό επαναλήψεων που θα τρέξει ο αλγόριθμος, την επιλογή του σχεδίου απόσβεσης, το ιστορικού κατανάλωσης ενέργειας ECP του χρήστη, τα k στοιχεία προς τροποποίηση και η θερμοκρασία T. Η λύση που θα επιστρέψει ο αλγόριθμος του ενεργειακού σχεδίου είναι ένας διάνυσμα s = < s1 , … , sn > μεγέθους N (N = αριθμός των Meta-rule). Αν Si = 0 τότε το Meta-rule παραβλέπετέ και αν Si = 1 τότε ο αλγόριθμος εκτελείται το Meta-rule.

Όταν ξεκινήσει ο αλγόριθμος όλα τα Meta-Rules τα οποία δίνει ο χρήστης εκτελούνται (Si = 1 για κάθε i) κάτι που είναι προς όφελος του χρήστη αλλά μπορεί να οδηγήσει με υψηλή πιθανότητα στο αποτέλεσμα να παραβιάζεται ο μέγιστος προϋπολογισμός έτσι όπως καθορίζεται από το σχέδιο απόσβεσης.

Ακολούθως ο αλγόριθμος προσπαθεί να βελτιστοποιήσει τα αποτελέσματα του χρησιμοποιώντας ένα simulated annealing heuristic, το οποίο δημιουργά μία καινούργια λύση. Αν η καινούργια λύση έχει καλύτερα αποτελέσματα, δηλαδή έχει μικρότερο κόστος σφάλματος άνεσης και μικρότερη κατανάλωση ενέργειας αποθηκεύεται.

Ο αλγόριθμος σταματά όταν περάσουν ένας αριθμός από επαναλήψεις ο οποίος δόθηκε από τον χρήστη ως είσοδος στον αλγόριθμο. Εναλλακτικά μπορεί να επαναλαμβάνεται μέχρι να μην υπάρχει κάποια καλύτερη λύση, όμως σε περίπτωση που δεν υπάρχει κάποια γνώση για την βέλτιστη λύση αυτό θα οδηγούσε σε άπειρη επανάληψη.

Algorithm 1 $IMCF$: generates an energy-efficient plan

Input: MRT: Meta-Rule Table; k: components to be modified; τ_{max}: max iterations; t: time granularity; apl: amortization plan; ECP: Energy Consumption Profile

Output: An energy plan solution $s^* = (s_1, \ldots, s_N)$

AP(apl, p, ECP) ▷ **Amortization Plan Routine**
  switch (apl)
  a: $E_p \leftarrow LAF(t, ECP)$ ▷ use linear Eq. (3)
  b: $E_p \leftarrow BLAF(t, ECP)$ ▷ use balloon Eq. (4)
  c: $E_p \leftarrow EAF(t, ECP)$ ▷ use ECP-based Eq. (5)

EP$(MRT, k, \tau_{max}, i, E_p)$ ▷ **Energy Plan Routine**
  $s^* \leftarrow init_i(MRT)$ ▷ s^*: initial solution for time i
  $(F_E, F_{CE}) \leftarrow evaluate(s^*)$ ▷ with Equations (1),(2)
  While $\tau < \tau_{max}$ **do** ▷ τ: current iteration
    $s \leftarrow optimization(s^*)$ ▷ randomly select k positions and swap their binary value
  $(F_E, F_{CE}) \leftarrow evaluate(s)$ ▷ with Equations (1),(2)
    If $(F_E(s) \leq E_p)$ && $(F_{CE}(s) < F_C(s^*))$ **then**
      $s^* \leftarrow s$ ▷ Set s as the current solution s^*
    EndIf
    $\tau++$ ▷ Increase iterations
  EndWhile
  return s^* ▷ Return the final energy plan solution

$apl \leftarrow AP(apl, t, ECP)$;
return $(\forall_i^t\ EP(MRT, k, \tau_{max}, i, apl))$

Σχήμα 3.1 Αλγόριθμος Energy Planner [45]

3.2.4 Αλγόριθμος Green Planner

Ο αλγόριθμος Green Planner είναι μία παραλλαγή του Energy Planner με την μόνη διαφορά ότι λαμβάνει υπόψιν και την εκπομπή ρύπων διοξειδίου του άνθρακα [46]. Δηλαδή για κάθε νέα λύση η οποία δημιουργείται ο έλεγχος γίνεται αν η καινούργια έχει μικρότερο comfort error, μικρότερη κατανάλωση ενέργειας και μικρότερη εκπομπή ρύπων και αποθηκεύεται. Επιπλέον ακόμα μία διαφορά με τον αλγόριθμο Energy Planner είναι ότι μπορεί να αποθηκευτεί η λύση ακόμα και αν είναι χειρότερα τα αποτελέσματα της με μία τυχαία πιθανότητα.

3.2.5 Προσεγγίσεις

Ο αλγόριθμος δοκιμάστηκε σε τρεις βασικές προσεγγίσεις. Αρχικά κανένας κανόνας δεν λαμβάνεται υπόψιν (NR) και έτσι έχουμε τα καλύτερα αποτελέσματα ως προς την εκπομπή ρύπων διοξειδίου του άνθρακα, αφού οι ρύποι ελαχιστοποιούνται αλλά παράλληλα έχουμε τα χειρότερα αποτελέσματα ως προς την άνεση του χρήστη αφού κανένας κανόνας που όρισε ο χρήστης δεν εκτελείται. Η δεύτερη προσέγγιση η οποία δοκιμάστηκε είναι να εκτελεστούν όλοι οι κανόνες (MR) οι οποίοι όρισε ο χρήστης. Σε αυτήν την περίπτωση έχουμε βέλτιστα αποτελέσματα ως προς την άνεση του χρήστη, όμως παράλληλα έχουμε τα χειρότερα αποτελέσματα όσων αφορά την εκπομπή ρύπων. Η IFTTT προσέγγιση, με δεδομένο την έλλειψη λεπτομερών κανόνων, θα ήταν μία αυθαίρετη ακολουθία εκτέλεσης κανόνων και τα αποτελέσματα είναι κάπου ανάμεσα στο NR και το MR. Ένα παράδειγμα ενός IFTTT κανόνα είναι «Αν είναι καλοκαίρι θέσε την θερμοκρασία στους 25 βαθμούς Κελσίου». Ο Energy Planner είναι μία πιο ολοκληρωμένη έκδοση των αυθαίρετων κανόνων του IFTTT. Ένα παράδειγμα ενός Meta-rule θα ήταν «Αν η ώρα είναι 10 π.μ. μέχρι 4 μ.μ. θέσε την θερμοκρασία στους 22 βαθμούς Κελσίου».

3.2.6 Αποτελέσματα

Έγιναν πειράματα για τρία διαφορετικά σε μέγεθος κτήρια, για μία οικία, για εστίες πανεπιστημίου και για ένα ξενοδοχείο. Τα αποτελέσματα έδειξαν ότι με τον Energy Planner μπορεί να υπάρξει μακροπρόθεσμη μείωση 10% στην κατανάλωση ενέργειας με μοναδικό αντάλλαγμα περίπου 2-4% στο κόστος σφάλματος άνεσης σε σχέση με τα αρχικά δεδομένα. Επιπλέον με την χρήση του Green Planner τα αποτελέσματα έδειξαν ότι μπορεί να υπάρξει μείωση 45-59% στην εκπομπή ρύπων διοξειδίου του άνθρακα με μοναδικό αντάλλαγμα περίπου 3% στο κόστος σφάλματος άνεσης [47].

Επίσης τα αποτελέσματα έδειξαν ότι ο Energy Planner είναι γρηγορότερος και πιο αποτελεσματικός σε σχέση με τις άλλες δοκιμές που έγινε και πιο συγκεκριμένα το NR, MR και IFTTT.

Κεφάλαιο 4

Αλγόριθμος GreenCap

Σε αυτό το κεφάλαιο θα αναλυθεί η δομή και η αρχιτεκτονική του αλγόριθμου GreenCap. Θα αναλυθεί πλήρως κάθε μέθοδος που χρησιμοποιείτε στον αλγόριθμο. Στο τέλος της ενότητας εμφανίζονται και κάποια παραδείγματα από το demo το οποίο υλοποιήθηκε.

4.1 Εισαγωγή

Ο αλγόριθμος GreenCap αναπτύχθηκε χρησιμοποιώντας την λογική του γενετικού αλγορίθμου. Ο λόγος που επιλέχτηκε ο γενετικός αλγόριθμος, έναντι αρχικά του simulated annealing του IMCF είναι ότι το IMCF έχει ως στόχο την μακροπρόθεσμη μείωση στην κατανάλωση ενέργειας, σε αντίθεση με το δικό μας πρόβλημα ότι έχουμε ως στόχο την βραχυπρόθεσμη μείωση. Επιπλέον το IMCF δεν λαμβάνει υπόψιν τις ώρες αιχμής, ούτε τις καιρικές συνθήκες και την παραγωγή ενέργειας από τις ανανεώσιμες πηγές ενέργειας. Ο αλγόριθμος ο οποίος αναπτύχθηκε και σε σχέση με το IMCF είναι δύο διαφορετικές προσεγγίσεις ενός παρόμοιου προβλήματος. Επιπλέον και με βάση την

μελέτη που έγινε σε σχετικά άρθρα που έχουν δημοσιευτεί, ο γενετικός αλγόριθμος μαζί με κάποιες επιπλέον ευρετικές συναρτήσεις μας δίνει την δυνατότητα να λαμβάνουμε υπόψιν, εκτός από την μείωση της κατανάλωσης ενέργειας, και επιπρόσθετα τις καιρικές συνθήκες που υπάρχουν, την παραγωγή ενέργειας από ανανεώσιμες πηγές ενέργειας και τις ώρες αιχμής της συνολικής ροής ενέργειας στο ενεργειακό δίκτυο. Σε αντίθεση με τους άλλους αλγορίθμους, ο γενετικός αλγόριθμος μας επιτρέπει επίσης να έχουμε πολλαπλά και συγκρουόμενα objectives, όπως για παράδειγμα την μείωση της κατανάλωσης ενέργειας, με την διατήρηση του επιπέδου άνεσης του χρήστη.

Ο αλγόριθμος GreenCap αποτελείται από τα εξής στάδια. Αρχικά γίνονται τα βασικά στάδια ενός γενετικού αλγορίθμου, δηλαδή η αρχικοποίηση του πληθυσμού, η επιλογή δύο τυχαίων λύσεων (selection), διασταύρωση (crossover), μετάλλαξη (mutation) και αξιολόγηση των δύο λύσεων με βάση την fitness function η οποία χρησιμοποιείτε. Επιπρόσθετα από τα βασικά βήματα ενός γενετικού αλγορίθμου, εκτελούνται ακόμα δύο ευρετικές συναρτήσεις. Η πρώτη ευρετική συνάρτηση η οποία εκτελείται, έχει ως στόχο την ανακατανομή της κατανάλωσης ενέργειας των έξυπνων συσκευών από τις ώρες αιχμής στις μη ώρες αιχμής. Η δεύτερη ευρετική συνάρτηση η οποία εκτελείται, έχει ως στόχο την επαναφορά της ημερήσιας κατανάλωσης ενέργειας στα αρχικά επίπεδα, όπου και αν υπάρχει αυξομείωση, η οποία μπορεί να προκλήθηκε από τις προηγούμενες συναρτήσεις. Γενικότερα οι δύο επιπρόσθετες συναρτήσεις εκτελούνται για να γίνει πιο σωστή εκμετάλλευση της παραγωγής των ανανεώσιμων πηγών ενέργειας, καθώς επίσης αποφεύγεται η επικόλληση σε πιθανόν τοπικών ελαχίστων κατά την διάρκεια εκτέλεσης του αλγορίθμου.

Ο συνδυασμός των συναρτήσεων αυτών αποτελεί την διεργασία διαχείρισης ενέργειας και το αποτέλεσμα τους είναι η χρονοδρομολόγηση της κατανάλωσης ενέργειας των έξυπνων οικιακών συσκευών. Περισσότερες λεπτομέρειες σχετικά με την λειτουργία της κάθε συνάρτησης περιγράφεται στα επόμενα υποκεφάλαια.

Το πρόβλημα που έχει ως στόχο να λύσει ο αλγόριθμος είναι αρχικά την χρονοδρομολόγηση της κατανάλωσης ενέργειας των έξυπνων οικιακών συσκευών με απώτερο στόχο την μείωση της εισαγόμενης ενέργειας από το ενεργειακό δίκτυο και παράλληλα την μείωση των εκπομπών ρύπων διοξειδίου του άνθρακα χωρίς να υπάρξει

συνολικά μείωση στην κατανάλωση ενέργειας στην οικία. Επιπλέον, ο αλγόριθμος επιχειρεί να μειώσει το κόστος της ενέργειας μέσω της μεταφοράς της κατανάλωσης ενέργειας από τις ώρες αιχμής στις μη ώρες αιχμής. Επίσης με την ανάπτυξη μίας επιπλέον έκδοσης του αλγορίθμου, ο χρήστης δίνοντας κάποιους κανόνες άνεσης στον αλγόριθμο, ο αλγόριθμος λαμβάνει υπόψιν τους κανόνες αυτούς. Ο χρήστης μπορεί να επιλέξει εάν θέλει ή όχι να λαμβάνονται υπόψιν οι κανόνες άνεσης του οποίους δίνει.

4.2 Γενετικός Αλγόριθμος

Ο γενετικός αλγόριθμος [36] είναι ευρετικός αλγόριθμος αναζήτησης ο οποίος είναι εμπνευσμένος από την θεωρία της φυσικής εξέλιξης του Charles Darwin. Ο αλγόριθμος αντικατοπτρίζει τη διαδικασία της φυσικής επιλογής όπου επιλέγονται τα πιο κατάλληλα άτομα για αναπαραγωγή προκειμένου να παραχθούν οι απόγονοι της επόμενης γενιάς.

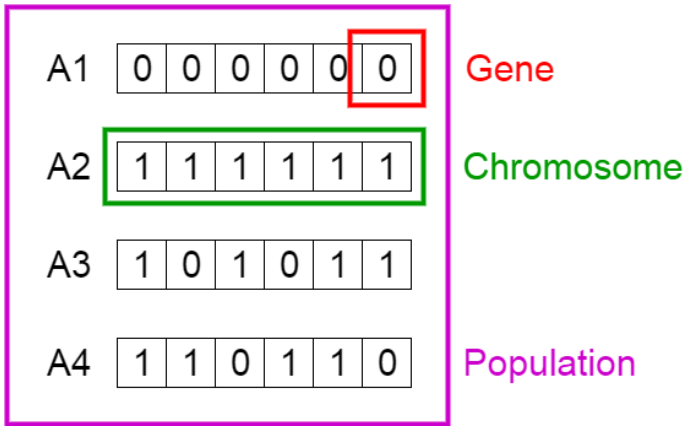
Η διαδικασία της φυσικής επιλογής ξεκινά με την επιλογή των πιο ικανών ατόμων από έναν πληθυσμό, τα οποία άτομα παράγουν απογόνους που κληρονομούν χαρακτηριστικά των γονέων και θα προστεθούν στην επόμενη γενιά. Εάν οι γονείς έχουν καλύτερη φυσική κατάσταση, οι απόγονοι τους θα είναι καλύτεροι από τους γονείς και θα έχουν περισσότερες πιθανότητες να επιβιώσουν. Αυτή η διαδικασία συνεχίζει να επαναλαμβάνεται και στο τέλος θα βρεθεί μια γενιά με τα καλύτερα άτομα.

Αυτή η φιλοσοφία του αλγορίθμου μπορεί να εφαρμοστεί σε ένα πρόβλημα αναζήτησης, εξετάζοντας ένα σύνολο λύσεων για ένα πρόβλημα και επιλέγουμε το σύνολο καλύτερων από αυτές.

Ο γενετικός αλγόριθμος αποτελείται από τις ακόλουθες 5 φάσεις.

4.2.1 Αρχικοποίηση Πληθυσμού (Initial Population)

Η διαδικασία ξεκινά με ένα σύνολο ατόμων που ονομάζεται πληθυσμός (Population). Κάθε άτομο είναι μία λύση στο πρόβλημα το οποίο θέλουμε να λύσουμε. Ένα άτομο χαρακτηρίζεται από ένα σύνολο παραμέτρων, γνωστών ως γονίδια (Genes). Τα γονίδια ενώνονται σε μία χορδή για να σχηματίσουν ένα χρωμόσωμα (Chromosome).



Σχήμα 4.1 Initial Population GA [39]

4.2.2 Συνάρτηση Καταλληλότητας (Fitness Function)

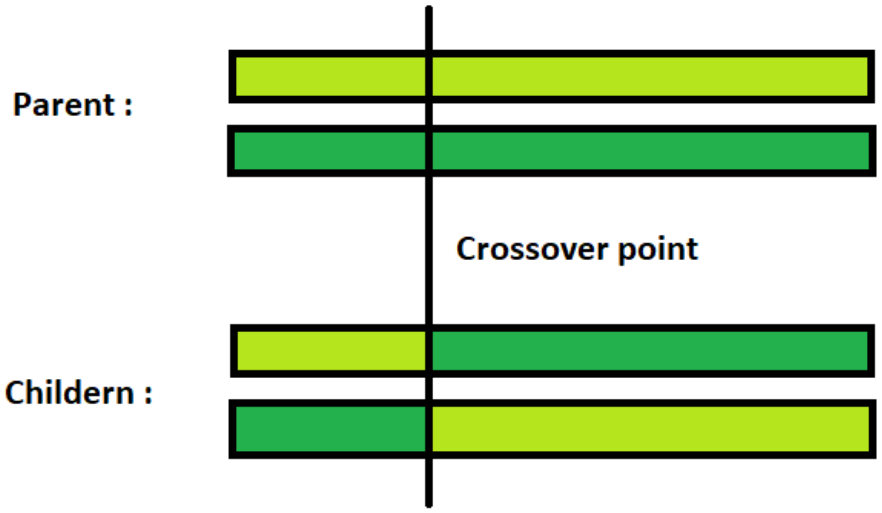
Η fitness function καθορίζει πόσο κατάλληλο είναι ένα άτομο. Δίνει βαθμολογία φυσικής κατάστασης σε κάθε άτομο στον πληθυσμό. Με πιο απλά λόγια είναι η συνάρτηση αξιολόγησης ενός ατόμου.

4.2.3 Επιλογή (Selection)

Η ιδέα της φάσης επιλογής στην δική μας περίπτωση είναι η τυχαία επιλογή δύο ατόμων, έτσι ώστε να περάσουν τα γονίδια τους στην επόμενη γενιά. Σε πιο γενική ιδέα ο γενετικός αλγόριθμος επιλέγει τα δύο πιο κατάλληλα γονίδια, όμως στην δική μας εκδοχή δεν ισχύει αυτό, καθώς επιλέγονται τυχαία δύο άτομα.

4.2.4 Διασταύρωση (Crossover)

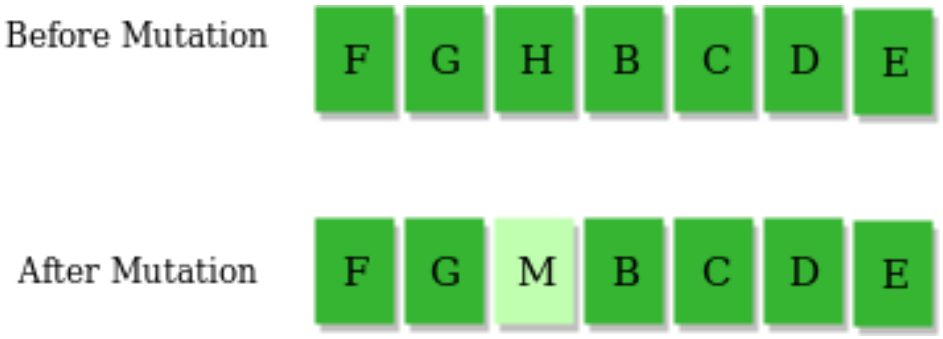
Το crossover (διασταύρωση) είναι η πιο σημαντική φάση σε ένα γενετικό αλγόριθμο. Για κάθε ζευγάρι γονέων που προκύπτει να ζευγαρώσουν, επιλέγεται τυχαία ένα σημείο διασταύρωσης (crossover point) μέσα από τα γονίδια. Οι απόγονοι δημιουργούνται ανταλλάσσοντας τα γονίδια των γονέων μεταξύ τους μέχρι το σημείο διασταύρωσης.



Σχήμα 4.2 Crossover GA [48]

4.2.5 Μετάλλαξη (Mutation)

Σε ορισμένους νέους απογόνους που σχηματίζονται, ορισμένα γονίδια τους μπορούν να υποβληθούν σε μετάλλαξη (mutation) με χαμηλή τυχαία πιθανότητα (στην δική μας περίπτωση 1%). Η μετάλλαξη συμβαίνει για να διατηρηθεί η ποικιλομορφία εντός του πληθυσμού και να αποτραπεί πρόωρη σύγκλιση.



Σχήμα 4.3 Mutation GA [49]

```
START
Generate the initial population
Compute fitness
REPEAT
    Selection
    Crossover
    Mutation
    Compute fitness
UNTIL population has converged
STOP
```

Σχήμα 4.4 Ψευδοκώδικας Γενετικού Αλγορίθμου [39]

4.3 Αλγόριθμος GreenCap

Σε αυτό το υποκεφάλαιο θα αναλυθεί ο αλγόριθμος GreenCap και των στοιχείων από τα οποία αποτελείται ο αλγόριθμος καθώς και τις παραμέτρους του.

Είσοδος: Ως είσοδος στον αλγόριθμο δίνονται τα δεδομένα με τις καταναλώσεις ενέργειας (σε kWh) των συσκευών κατά την διάρκεια ενός έτους (τα δεδομένα είναι μετρήσεις ανά ώρα), τα δεδομένα παραγωγής ενέργειας από το φωτοβολταϊκό σύστημα του σπιτιού για την ίδια χρονική περίοδο, τα δεδομένα που αφορούν την συνολική ροή ενέργειας στο ενεργειακό δίκτυο για να βρεθούν οι ώρες αιχμής, η μέγιστη κατανάλωση που μπορεί να κάνει μία συσκευή, το μέγεθος του πληθυσμού, δηλαδή τον αριθμό των τυχαίων λύσεων που θα δημιουργηθούν χρησιμοποιώντας τον αλγόριθμο Random όπως αυτός περιγράφεται στο Κεφάλαιο 5.2.2 και ο μέγιστος αριθμός επαναλήψεων (γενιές) που θα τρέξει ο γενετικός αλγόριθμος.

Αναπαράσταση λύσης: Ο αλγόριθμος θα επιστρέψει μία λίστα μεγέθους N (όπου N είναι οι συνολικές ώρες που υπάρχουν κατά την διάρκεια ενός έτους). Κάθε στοιχείο της λίστας είναι ένας πίνακας όπου περιέχει την κατανάλωση (σε kWh) που θα γίνει από τις

συσκευές στην συγκεκριμένη ώρα της ημέρας (η ώρα και η ημερομηνία βρίσκονται στην θέση 0 του πίνακα), και επίσης την παραγωγή που έγινε την συγκεκριμένη ώρα.

Αρχικοποίηση πληθυσμού: Στην αρχή του αλγορίθμου, το πρώτο βήμα το οποίο εκτελείται είναι η αρχικοποίηση του πληθυσμού του. Ο αλγόριθμος δημιουργεί p (μέγεθος πληθυσμού) τυχαίες λύσεις από τα δεδομένα που δίνονται. Στην συνέχεια, για κάθε λύση η οποία δημιουργήθηκε γίνονται κάποιες μετρήσεις.

Κατά την διάρκεια αρχικοποίησης του αλγορίθμου υπολογίζεται επίσης η κατανάλωση ενέργειας που έγινε από τις συσκευές, για κάθε μέρα που υπάρχει μέσα στα δεδομένα, όπου αυτό θα χρησιμοποιηθεί αργότερα στην συνάρτηση διόρθωσης της κατανάλωσης ενέργειας.

Βελτιστοποίηση: Στόχος του βελτιστοποίησης του αλγορίθμου είναι η ελαχιστοποίηση του fitness function, όπου αυτό περιγράφεται στο βήμα αξιολόγησης. Για το βήμα βελτιστοποίησης εκτελούνται συνολικά 6 συναρτήσεις του αλγορίθμου. Αρχικά γίνεται επιλογή 2 τυχαίων λύσεων μέσα από τον πληθυσμό ο οποίος δημιουργήθηκε (selection). Ακολούθως εκτελείται η συνάρτηση διασταύρωσης των δύο λύσεων (crossover). Η συνάρτηση crossover εκτελείται για κάθε μέρα μέσα στα δεδομένα, επιλέγεται ένα τυχαίο σημείο μέσα σε κάθε μέρα (από 0 μέχρι το 23), και με πιθανότητα 90% γίνεται ανταλλαγή των δεδομένων των δύο λύσεων (offspring) μέχρι σε εκείνο το σημείο.

Παρόμοια και η συνάρτηση μετάλλαξης (mutation), εκτελείται για κάθε μέρα μέσα στα δεδομένα, όπου επιλέγεται ένα τυχαίο σημείο μέσα σε κάθε μέρα (από 0 μέχρι το 23), και με πιθανότητα 1% σε αυτήν την ώρα θα κλείσουν όλες οι συσκευές, δηλαδή η κατανάλωση ενέργειας θα γίνει ίση με 0 kWh.

Ακολούθως θα εκτελεστεί και στις δύο λύσεις η συνάρτηση επιδιόρθωσης κατανάλωσης ενέργειας. Αναλυτικότερα, για κάθε μέρα μέσα στα δεδομένα και για κάθε συσκευή, αν η κατανάλωση της συσκευής μέσα στην συγκεκριμένη μέρα είναι μεγαλύτερη από την αρχική, θα μειώσει την κατανάλωση ενέργειας με άπληστο τρόπο, δηλαδή θα μειώσει την κατανάλωση ενέργειας ξεκινώντας από τις ώρες που δεν υπάρχει παραγωγή ενέργειας. Αν η κατανάλωση ενέργειας είναι μικρότερη από την αρχική, τότε θα αυξήσει

την κατανάλωση ενέργειας ξανά με άπληστο τρόπο, δηλαδή θα αυξήσει την κατανάλωση ενέργειας ξεκινώντας από τις ώρες που υπάρχει μεγαλύτερη παραγωγή ενέργειας.

Στην συνέχεια θα αξιολογηθούν οι δύο λύσεις χρησιμοποιώντας το fitness function, και θα επιλεγεί ο καλύτερη λύση, δηλαδή η λύση με το μικρότερο fitness. Σε αυτήν την λύση θα εκτελεστεί η συνάρτηση ανακατανομή ενέργειας από τις τρεις ώρες αιχμή στις τρεις μη ώρες αιχμής. Πιο συγκεκριμένα για κάθε μέρα μέσα στα δεδομένα θα βρεθούν οι τρεις ώρες αιχμής και οι τρεις μη ώρες αιχμής από τα δεδομένα της συνολικής ροής στο ενεργειακό δίκτυο. Ακολούθως θα γίνει η ανταλλαγή των καταναλώσεων, με προϋπόθεση ότι η κατανάλωση στις ώρες αιχμής είναι μεγαλύτερη από την κατανάλωση στις μη ώρες αιχμής. Αν η ανταλλαγή αυτή έχει λιγότερη εισαγόμενη ενέργειας από το ενεργειακό δίκτυο αποθηκεύεται αλλιώς παραβλέπετε.

Αξιολόγηση: Η συνάρτηση αξιολόγησης η οποία εκτελείται, επιλέγεται από τον χρήστη. Αναλυτικότερα, αν ο χρήστης επιλέξει ότι θέλει να λαμβάνονται υπόψιν οι κανόνες άνεσης τους οποίους έδωσε θα εκτελεστεί η αξιολόγηση GreenCapComfort, αλλιώς θα εκτελεστεί η αξιολόγηση GreenCap, όπως αυτές περιγράφονται παρακάτω.

Αξιολόγηση GreenCap: Μετά την εκτέλεση των προηγούμενων συναρτήσεων παράχθηκαν 3 λύσεις. Η αξιολόγηση της κάθε λύσης γίνεται με βάση την εισαγόμενη ενέργεια από το ενεργειακό δίκτυο, και άρα παράλληλα και στην εκπομπή ρύπων διοξειδίου του άνθρακα. Η λύση με την μικρότερη εισαγωγή ενέργειας είναι και η καλύτερη λύση μεταξύ των 3 λύσεων οι οποίες παράχθηκαν. Η συγκεκριμένη λύση τοποθετείτε στον πληθυσμό του γενετικού αλγορίθμου στην θέση της χειρότερης λύσης, δηλαδή της λύσης με την μεγαλύτερη εισαγωγή ενέργειας. Με αυτό τον τρόπο αποφεύγεται η πιθανότητα να χαθεί κάποια λύση όπου το αποτέλεσμα της είναι καλύτερο από οποιαδήποτε άλλη λύση.

Αξιολόγηση GreenCapComfort: Εναλλακτικά η αξιολόγηση του αλγορίθμου μπορεί να γίνει λαμβάνοντας υπόψιν και την άνεση του χρήστη. Ο χρήστης μπορεί να δώσει κάποιους κανόνες (π.χ. Να δουλέψει ο θερμοσίφωνας η ώρα 16:00 στις 21/10), και υπολογίζεται μία επιπλέον μετρική, το κόστος σφάλματος άνεσης του χρήστη. Ακολούθως στην fitness function λαμβάνονται υπόψιν και η εισαγομένη ενέργεια από το

ενεργειακό δίκτυο, και το κόστος σφάλματος άνεσης του χρήστη. Αναλυτικότερα λαμβάνεται υπόψιν κατά 75% η εισαγόμενη ενέργεια από το δίκτυο και κατά 25% το κόστος σφάλματος άνεσης του χρήστη.

Τερματισμός: Ο αλγόριθμος τερματίζει με την πάροδο g_{max} γενεών. Σε κάθε γενιά ο αλγόριθμος επαναλαμβάνει την διαδικασία όπως περιεγράφηκε προηγουμένως όσες φορές είναι το μέγεθος του πληθυσμού έχοντας ως στόχο την αναπλήρωση του παρόντος πληθυσμού με καινούργιες λύσεις με καλύτερα αποτελέσματα από την προηγούμενη γενιά.

Εναλλακτικά θα μπορούσε ο αλγόριθμος να τερματίσει μέχρι να βρεθεί μία λύση όπου το fitness της λύσης να είναι ίση με ένα αριθμό ο (=optimal), όμως αυτό μπορεί να μας οδηγούσε σε infinite loop, καθώς θα μπορούσε να τρέχει για πάντα και να μην μπορέσει ο αλγόριθμος να βρει την λύση που επιθυμούμε.

Παράδειγμα: Θεωρούμε ένα σύνολο από πραγματικά δεδομένα που πάρθηκαν από μία οικία στις ΗΠΑ, και αφορούν την κατανάλωση ενέργειας από διάφορες συσκευές μέσα στην οικία. Επίσης θεωρούμε ένα σύνολο από δεδομένα που αφορούν την παραγωγή ενέργειας που έγινε από το φωτοβολταϊκό σύστημα του ίδιου σπιτιού. Ακόμη, ένα σύνολο από δεδομένα που αφορούν την συνολική κατανάλωση ενέργειας που γίνεται στο ενεργειακό δίκτυο της πολιτείας που βρίσκεται το σπίτι. Επιπλέον ένα σύνολο από κανόνες που έδωσε ο ιδιοκτήτης της οικίας, για το πότε θέλει να λειτουργούν κάποιες από τις συσκευές στο σπίτι. Ο χρήστης θα βάλει σε λειτουργία τον αλγόριθμο επιλέγοντας την έκδοση του αλγορίθμου την οποία θέλει και ο αλγόριθμος θα δημιουργήσει μία χρονοδρομολόγηση της κατανάλωσης των συσκευών, όπου η εισαγόμενη ενέργειας από το ενεργειακό δίκτυο και παράλληλα οι εκπομπές διοξειδίου του άνθρακα θα ελαχιστοποιηθούν. Αν ο χρήστης επιλέξει την έκδοση του αλγόριθμου όπου λαμβάνεται υπόψιν και η άνεση του χρήστη, τότε τα αποτελέσματα του αλγορίθμου θα έχουν και αρκετά ψηλά το επίπεδο άνεσης του χρήστη.

Στο παρακάτω σχήμα φαίνεται ο ψευδοκώδικας του αλγορίθμου GreenCap. Ανάλογα με την έκδοση που επιλέγει ο χρήστης (μεταβλητή *ev*) εκτελείται η κατάλληλη συνάρτηση αξιολόγησης.

Algorithm 1 $GreenCap$: generates an energy-efficient plan

Input: ECT: Energy Consumption Table; EPT: Energy Production Table; GDT: Grid Demand Table; RT: Rule Table; g_{max}: max generation; p: population size; ev: Evaluation Function

Output: An energy plan solution ECT^*

GreenCap(ECT, p, g_{max}, ev) ▷ **GreenCap Routine**
  $dcpa \leftarrow findDailyConsumption(ECT)$ ▷ $dcpa$: daily consumption per appliance
  $pop \leftarrow population(ECT, p)$ ▷ pop: init p random solutions
  $calculateFitness()$ ▷ calculate fitness for each chromosome in population
  While $g < gmax$ **do** ▷ g: current generation
    $i = 0$
    While $i < p$ **do** ▷ i: current iteration
      $(O_1, O_2) \leftarrow selection()$ ▷ O_1, O_2: random selected from pop
      $crossover(O_1, O_2)$ ▷ crossover at a random point with 90% chance
      $mutation(O_1, O_2)$ ▷ mutation at a random point with 1% chance
      $fixConsumption(O_1, O_2, dcpa)$ ▷ fix consumption of offsprings
      $(F_1, F_2) \leftarrow evaluate(O_1, O_2, EPT, ev)$ ▷ F_1, F_2: fitness of O_1, O_2
      $O_3 \leftarrow fittest(F_1, F_2)$ ▷ O_3: fittest offspring between O_1, O_2
      $peakHourReallocation(O_3, GDT)$ ▷ peak to non-peak reallocation
      $F_3 \leftarrow evaluate(O_3, EPT, ev)$ ▷ F_3: fitness of O_3
      $O \leftarrow fittest(F_1, F_2, F_3)$ ▷ O: fittest offspring between O_1, O_2, O_3
      $populate(O)$ ▷ populate fittest offspring to least fit in pop
      $i++$ ▷ increase iterations
    EndWhile
    $g++$ ▷ increase generations
  EndWhile

$ECT^* \leftarrow fittest(pop)$; ▷ ECT^*: fittest chromosome in population
return (ECT^*)

Σχήμα 4.5 Αλγόριθμος GreenCap

4.4 Εφαρμογή (Demo)

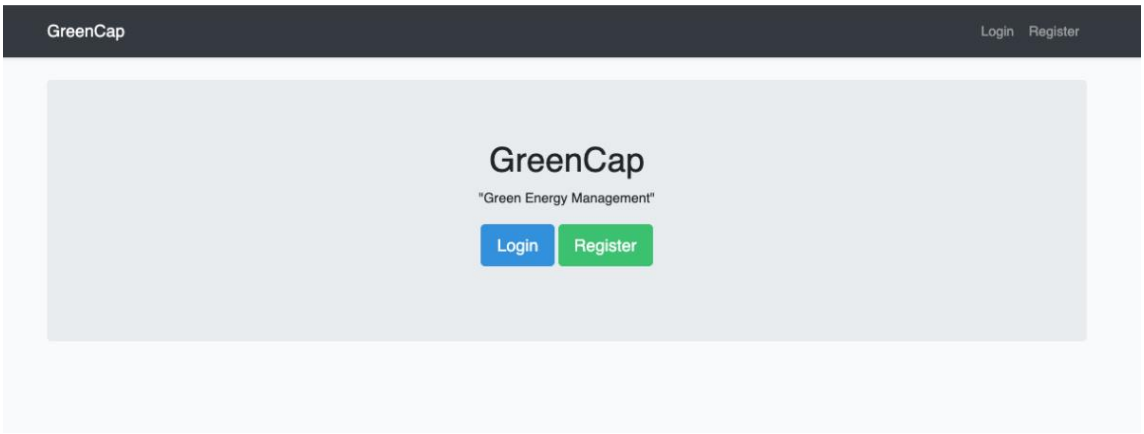
Στα πλαίσια της Ατομικής Διπλωματικής Εργασίας μου, υλοποιήθηκε και ένα demo όπου ο χρήστης μπορεί να δημιουργήσει λογαριασμό στο σύστημα, να εισάγει τις συσκευές που έχει στην οικία του και να δημιουργήσει κάποιους κανόνες λειτουργίας όπως ορίστηκαν προηγουμένως.

Ο αλγόριθμος θα διαβάσει από την βάση δεδομένων όπου θα φυλαχτούν οι κανόνες και οι συσκευές και ανάλογα θα του δημιουργήσει ένα πρόγραμμα που θα λειτουργούν οι

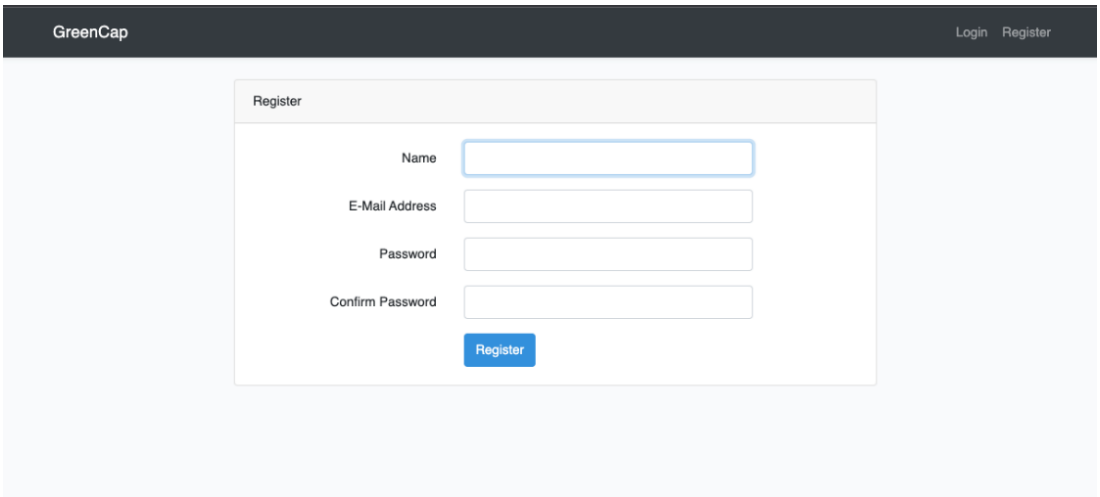
συσκευές, έτσι ώστε να έχει όσων το δυνατών λιγότερη εισαγωγή ενέργειας και παράλληλα εκπομπές ρύπων CO2, κρατώντας την άνεση του χρήστη σε ψηλά επίπεδα.

Για την ανάπτυξη του demo (Graphical User Interface) χρησιμοποιήθηκε to Laravel PHP framework, ακολουθώντας την αρχιτεκτονική Model-View-Controller, καθώς επίσης και JavaScript και HTML. Για την εκτέλεση του κώδικα του demo χρησιμοποιήθηκε ο NGINX web-server. Επίσης η βάση η οποία χρησιμοποιήθηκε είναι MariaDB.

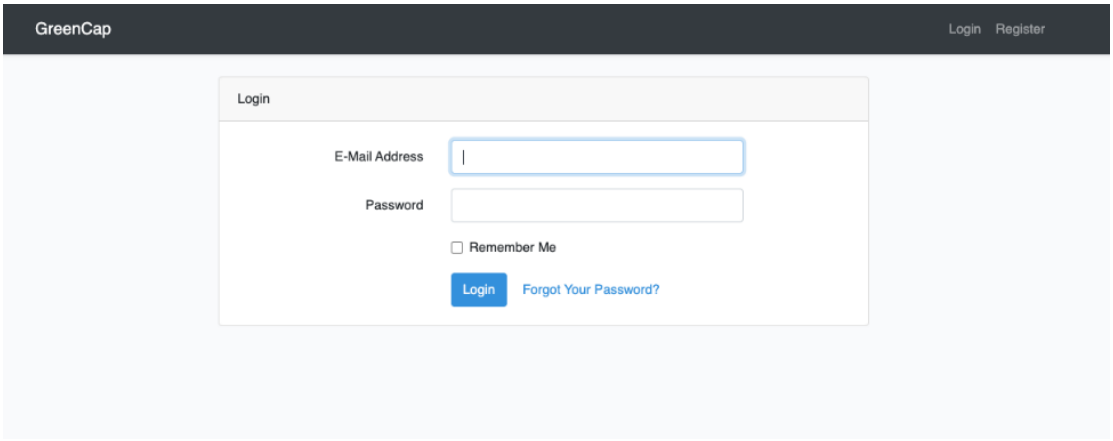
Παρακάτω ακολουθούν κάποια σχήματα όπου φαίνεται το demo το οποίο αναπτύχθηκε για την ευκολότερη διάδραση του χρήστη με τον αλγόριθμο.



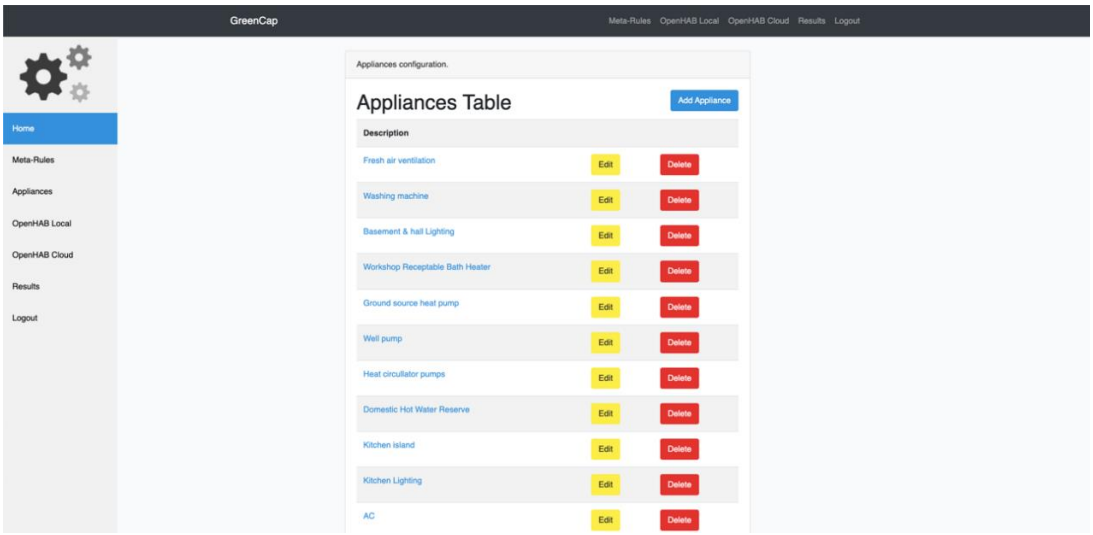
Σχήμα 4.6 Αρχική σελίδα demo



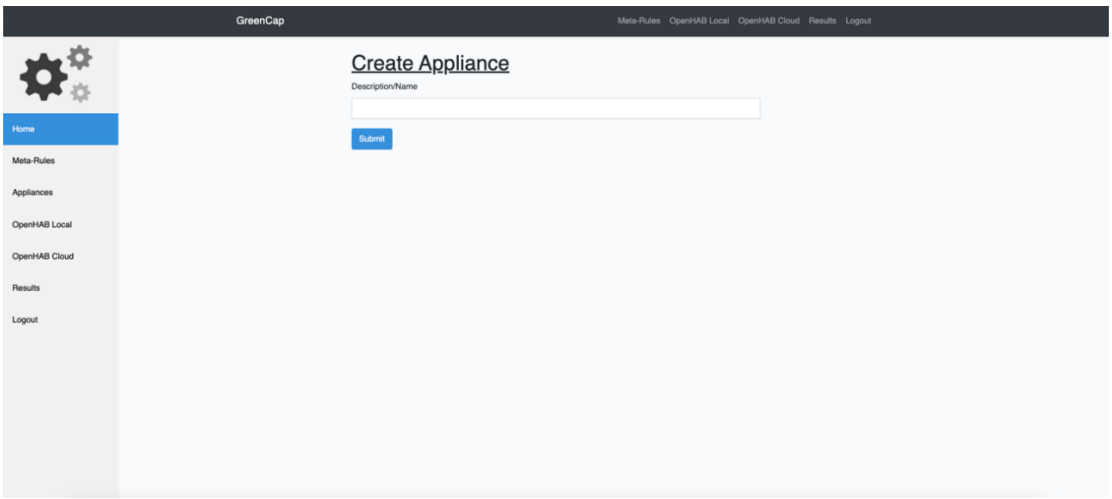
Σχήμα 4.7 Σελίδα εγγραφής



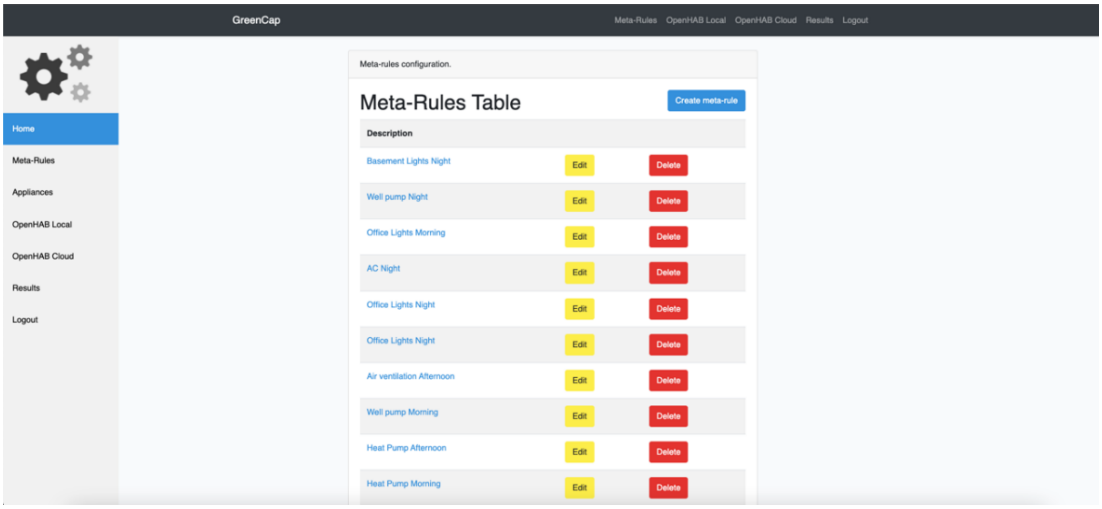
Σχήμα 4.8 Σελίδα εισόδου



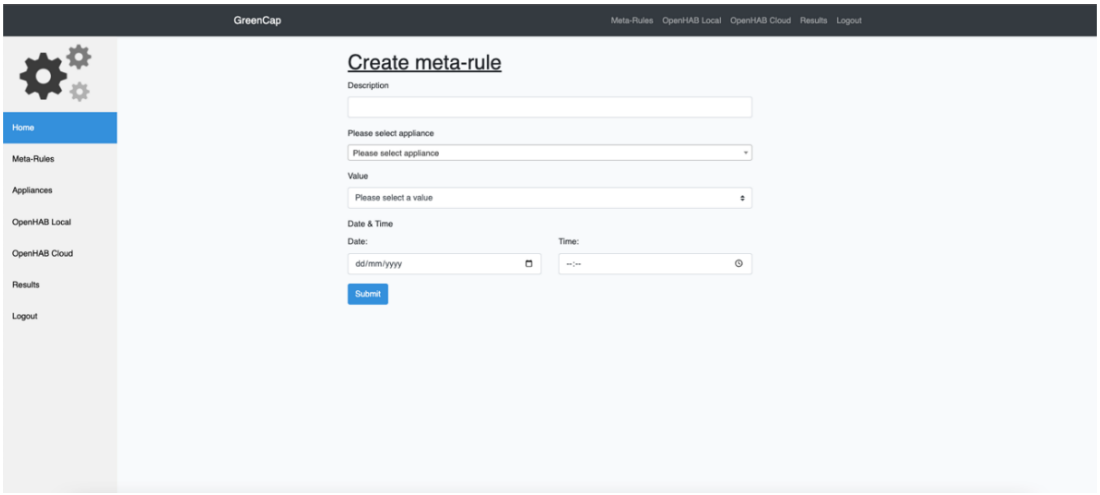
Σχήμα 4.9 Σελίδα προβολής συσκευών



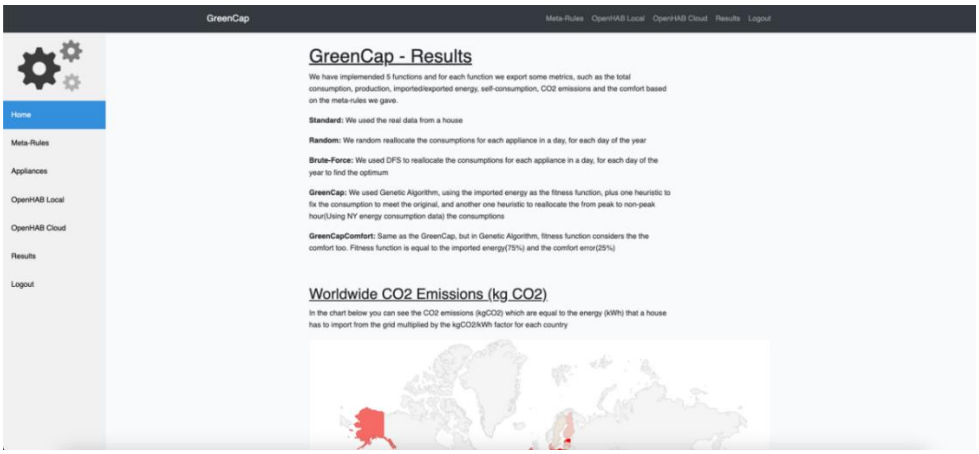
Σχήμα 4.10 Σελίδα δημιουργία συσκευής



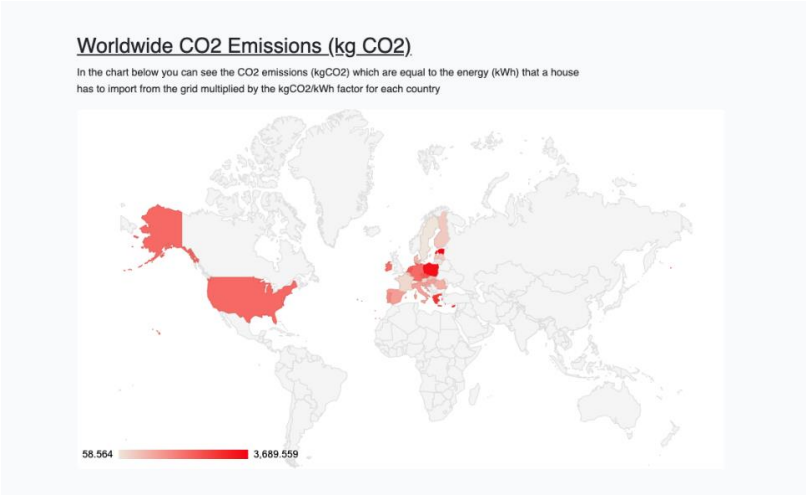
Σχήμα 4.11 Σελίδα προβολής κανόνων άνεσης



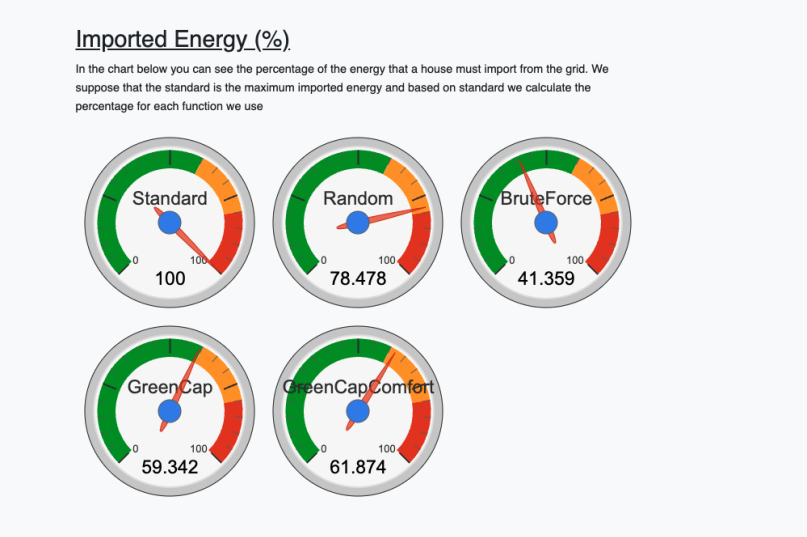
Σχήμα 4.12 Σελίδα δημιουργίας κανόνων άνεσης



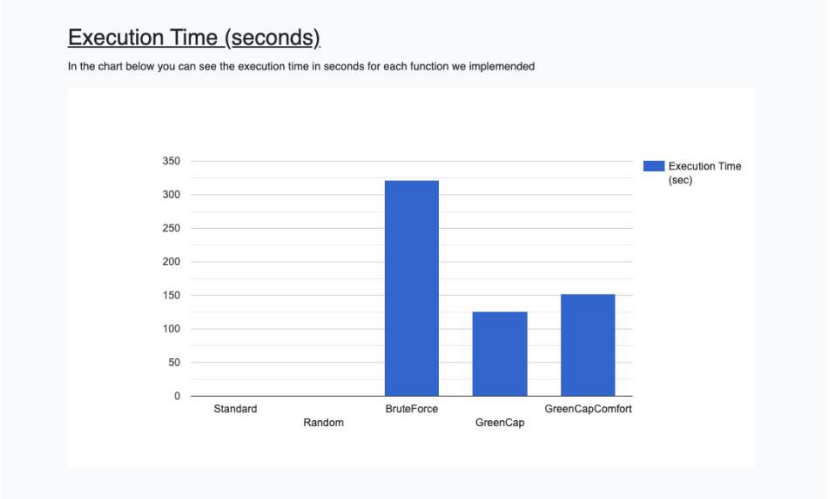
Σχήμα 4.13 Σελίδα παρουσίασης αποτελεσμάτων 1



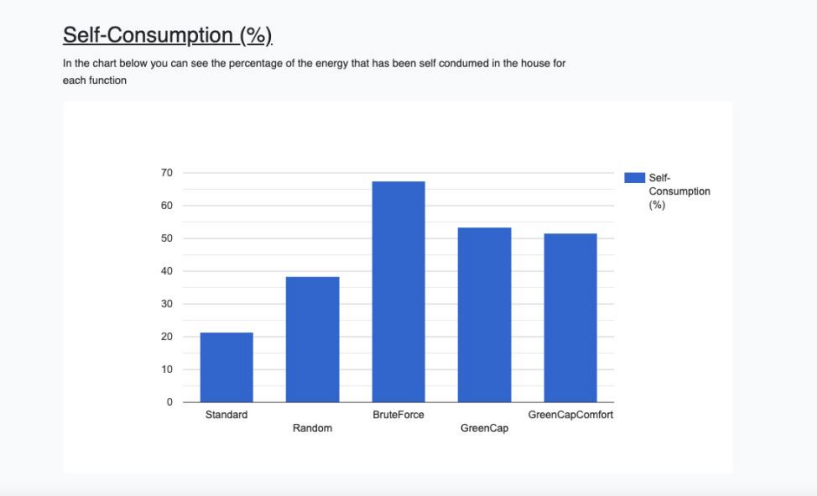
Σχήμα 4.14 Σελίδα παρουσίασης αποτελεσμάτων 2



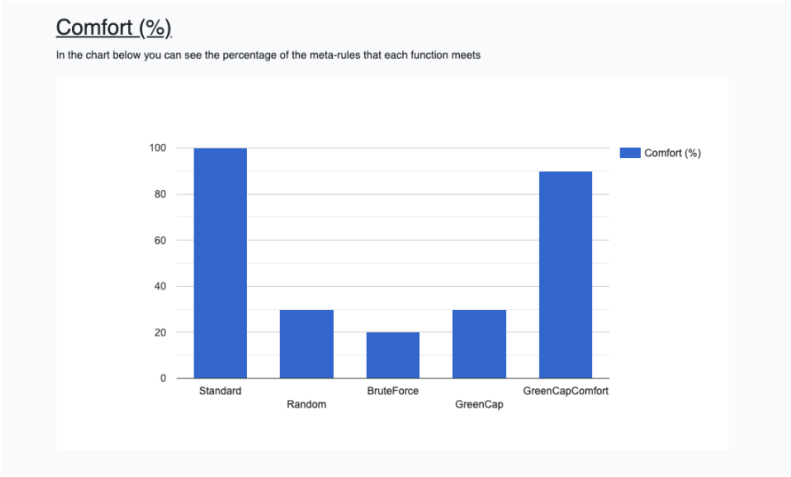
Σχήμα 4.15 Σελίδα παρουσίασης αποτελεσμάτων 3



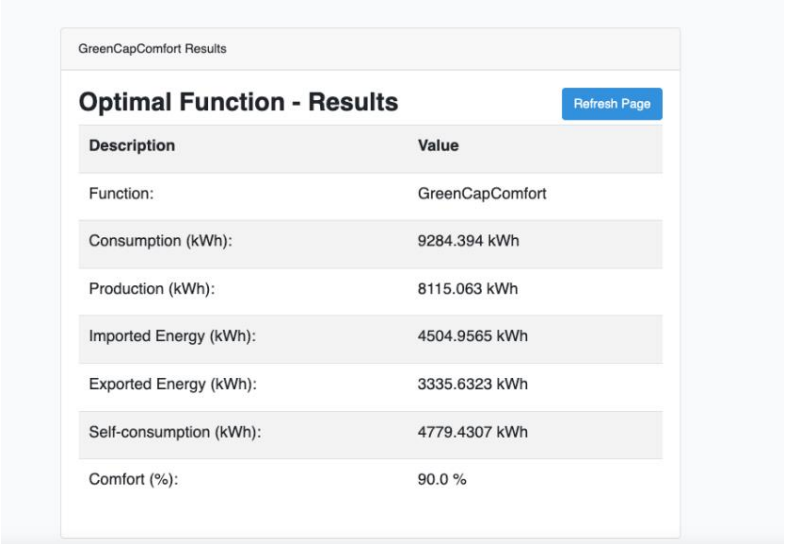
Σχήμα 4.16 Σελίδα παρουσίασης αποτελεσμάτων 4



Σχήμα 4.17 Σελίδα παρουσίασης αποτελεσμάτων 5



Σχήμα 4.18 Σελίδα παρουσίασης αποτελεσμάτων 6



Σχήμα 4.19 Σελίδα παρουσίασης αποτελεσμάτων 7

Κεφάλαιο 5

Πειραματική Μεθοδολογία και Αποτίμηση

Σε αυτό το κεφάλαιο αναλύεται η πειραματική μεθοδολογία που χρησιμοποιήθηκε και η αποτίμηση των διαφόρων αλγορίθμων των οποίων υλοποιήθηκαν. Θα αναφερθούν όλα τα δεδομένα, οι μετρικές καθώς και οι αλγόριθμοι οι οποίοι χρησιμοποιήθηκαν, και ακολούθως θα αναλυθεί η πειραματική αξιολόγηση η οποία έγινε.

5.1 Πειραματική Μεθοδολογία

Σε αυτήν το υποκεφάλαιο θα αναφερθούν περισσότερες λεπτομέρειες σχετικά με την μεθοδολογία η οποία ακολουθήθηκε για την σχεδίαση και ανάπτυξη των αλγορίθμων οι οποίοι υλοποιήθηκαν. Θα αναφερθεί η πλατφόρμα στην οποία έγιναν οι δοκιμές, τα δεδομένα, οι μετρικές και οι αλγόριθμοι που χρησιμοποιήθηκαν κατά τις δοκιμές τις οποίες κάναμε.

5.1.1 Πλατφόρμα

Η αξιολόγηση πραγματοποιήθηκε στο κέντρο δεδομένων DMSL laboratory το οποίο είναι ένα ιδιωτικό κέντρο δεδομένων. Ο υπολογιστικός μας κόμβος έχει ως λειτουργικό σύστημα το Ubuntu 18.04 server image, έχει 8GB RAM και δυο νοητές CPU των 2.40GHz. Η εικόνα χρησιμοποιεί τοπικά 10K RPM RAID-5 LSILogic SCSI δίσκους, μορφοποιημένοι με VMFS 6 (1MB block size).

5.1.2 Δεδομένα

Έχουμε υιοθετήσει μία πειραματική μεθοδολογία που βασίζεται σε ίχνη από πραγματικά σύνολα δεδομένων τα οποία τροφοδοτούνται στον προσομοιωτή μας ο οποίος εκτελείται στην πλατφόρμα μας. Αυτό μας επιτρέπει να εκτελούμε επαναλαμβανόμενα, με την ευχέρεια να μπορούμε απλώς να αλλάζουμε τις παραμέτρους τις οποίες έχουμε θέσει. Αναλυτικότερα, χρησιμοποιήθηκαν 3 διαφορετικά σύνολα δεδομένων από πραγματικές μετρήσεις οι οποίες έγιναν. Το πρώτο σύνολο δεδομένων είναι μετρήσεις οι οποίες έγιναν σε μία οικία, όπου μετρήθηκαν οι καταναλώσεις ενέργειας οι οποίες έγιναν από διάφορες οικιακές συσκευές, όπως φούρνο, πλυντήριο κτλ. Οι μετρήσεις έγιναν από το Laboratory for Advance System Software (LASS) το οποίο βρίσκεται κάτω από το University of Massachusetts Amherst. Οι μετρήσεις πάρθηκαν στα πλαίσια του Smart*: Optimizing Energy Consumption in Smart Homes project το οποίο είχε ως στόχο να βελτιστοποιήσει την κατανάλωση ενέργειας στο σπίτι. Πιο συγκεκριμένα πάρθηκαν μετρήσεις για την κατανάλωση ενέργειας από διάφορες έξυπνες συσκευές σε διάφορα σπίτια, καιρικές συνθήκες στα σημεία όπου βρίσκονται τα σπίτια καθώς και μετρήσεις από διάφορα φωτοβολταϊκά συστήματα και την παραγωγή ηλιακής ενέργειας από αυτά [50]. Επίσης χρησιμοποιήθηκε ακόμα ένα σύνολο δεδομένων το οποίο πάρθηκε από την διαχείριση

ενεργειακών πληροφοριών των Ηνωμένων Πολιτειών της Αμερικής, το οποίο έχει τις μετρήσεις από την συνολική ροή ενέργειας η οποία μεταφέρεται στο ενεργειακό δίκτυο για την κάλυψη των ενεργειακών αναγκών των Ηνωμένων Πολιτειών της Αμερικής, το οποίο χρησιμοποιήθηκε για την εύρεση των ωρών αιχμής [51].

Σύνολο δεδομένων Κατανάλωσης Ενέργειας σε οικία

Το σύνολο των δεδομένων είναι μεγέθους 408MB και αποτελείται από 527.040 μετρήσεις (γραμμές δεδομένων) ανά λεπτό από την 01/01/2016 00:00 μέχρι τις 31/12/2016 23:59. Το σύνολο αποτελείται από 20 στήλες, όπου η πρώτη είναι η ώρα και ημερομηνία και οι υπόλοιπες 19 στήλες αποτελούν μετρήσεις κατανάλωσης ενέργειας από 19 διαφορετικές οικιακές συσκευές σε kWh.

Τα δεδομένα καταγράφηκαν από διάφορους μετρητές μέσα σε 1 οικία. Έγινε κάποια προεπεξεργασία έτσι ώστε να αναπληρωθούν κάποιες χαμένες μετρήσεις, να συνδυαστούν τα σύνολα των δεδομένων σε ένα σύνολο, και να αφαιρεθούν κάποιες αχρείαστες μετρήσεις έτσι ώστε να έχουμε ένα ολοκληρωμένο σύνολο από δεδομένα. Στον Πίνακα 5.1 φαίνεται ένα παράδειγμα από τα δεδομένα της κατανάλωσης ενέργειας.

Date & Time	Appliance 1 (kW)	Appliance 2 (kW)
01/01/2016 0:00	0	0
01/01/2016 0:01	0	0
01/01/2016 0:02	0	0
01/01/2016 0:03	0	0
01/01/2016 0:04	0	0
01/01/2016 0:05	0.001456	0
01/01/2016 0:06	0	0
01/01/2016 0:07	0	0
01/01/2016 0:08	0	0
01/01/2016 0:09	0	0
01/01/2016 0:10	0	1.67E-05
01/01/2016 0:11	0.001482	0
01/01/2016 0:12	0	0
01/01/2016 0:13	0	0

Πίνακας 5.1 Παράδειγμα δεδομένων κατανάλωση ενέργειας ανά λεπτό

Σύνολο δεδομένων Παραγωγής Ενέργειας από Φωτοβολταϊκό Σύστημα

Τα σύνολο δεδομένων το οποίο χρησιμοποιήθηκε για την παραγωγή ενέργειας από ένα φωτοβολταϊκό σύστημα έχει μέγεθος 3.5 MB και αποτελείται από 65741 μετρήσεις (γραμμές δεδομένων) ανά ώρα από την 30/12/2010 05:00 μέχρι τις 30/16/2017 16:00. Το σύνολο δεδομένων αποτελείται από 2 στήλες, όπου η πρώτη είναι η ώρα και ημερομηνία και η δεύτερη στήλη η παραγωγή ενέργειας η οποία έγινε σε kWh. Το παρών φωτοβολταϊκό σύστημα είναι μεγέθους 5.5 kWh, δηλαδή η μέγιστη παραγωγή που μπορεί να κάνει ανά ώρα είναι 5.5 kWh.

Και σε αυτό το σύνολο δεδομένων έγινε κάποια προεπεξεργασία για την αφαίρεση των αχρείαστων γραμμών δεδομένων, δηλαδή να μείνουν μόνο οι γραμμές δεδομένων που αφορούν την χρονιά 2016, έτσι ώστε να ταιριάζουν με τα προηγούμενα δεδομένα.

Date & Time	Solar energy generated (kWh)
01/01/2016 0:00	0
01/01/2016 1:00	0
01/01/2016 2:00	0
01/01/2016 3:00	0
01/01/2016 4:00	0
01/01/2016 5:00	0
01/01/2016 6:00	0
01/01/2016 7:00	0
01/01/2016 8:00	0
01/01/2016 9:00	0.11970806
01/01/2016 10:00	0.320995
01/01/2016 11:00	0.52346389
01/01/2016 12:00	2.27059667
01/01/2016 13:00	3.68024111
01/01/2016 14:00	3.42192694

Πίνακας 5.2 Παράδειγμα δεδομένων παραγωγής ενέργειας ανά ώρα

Σύνολο δεδομένων από Συνολική Ροή Ενέργειας στις ΗΠΑ

Το σύνολο δεδομένων το οποίο χρησιμοποιήθηκε για την εύρεση των ωρών αιχμής στην κατανάλωση ενέργειας στις ΗΠΑ είναι μεγέθους 63.1 MB και αποτελείται από 579746 μετρήσεις (γραμμές δεδομένων) ανά ώρα από διάφορους οργανισμούς ενέργειας σε όλες

τις πολιτείες της Αμερικής από 01/01/2016 10:00 μέχρι τις 31/12/2016 23:00. Το σύνολο αποτελείται από 15 στήλες.

Και σε αυτό το σύνολο δεδομένων έγινε προεπεξεργασία, και πιο συγκεκριμένα κρατήθηκε μόνο ένας οργανισμός ενέργειας, αυτός της Νέας Υόρκης, και αφαιρέθηκαν και αρκετές στήλες. Κρατήθηκαν μόνο 3 στήλες, η πρώτη είναι το όνομα του οργανισμού, η δεύτερη στήλη αποτελείται από την ημερομηνία και την ώρα, και στην τρίτη στήλη βρίσκεται η συνολική κατανάλωση ενέργειας σε mW.

Balancing Authority	Local Time at End of Hour	Demand (mW)
NYIS	01/01/2016 1:00	15,250
NYIS	01/01/2016 2:00	14,583
NYIS	01/01/2016 3:00	14,013
NYIS	01/01/2016 4:00	13,632
NYIS	01/01/2016 5:00	13,500
NYIS	01/01/2016 6:00	13,671
NYIS	01/01/2016 7:00	14,093
NYIS	01/01/2016 8:00	14,398
NYIS	01/01/2016 9:00	14,791
NYIS	01/01/2016 10:00	15,502

Πίνακας 5.3 Παράδειγμα δεδομένων ροής ενέργειας στο ενεργειακό δίκτυο ανά ώρα

5.1.3 Μετρικές

Οι μετρικές οι οποίες χρησιμοποιήθηκαν για την αξιολόγηση των αποτελεσμάτων αποτελούνται από την συνολική κατανάλωση ενέργειας, την συνολική παραγωγή ενέργειας, την ποσότητα ενέργειας που πρέπει να εισαχθεί από το δίκτυο (imported energy), την ποσότητα ενέργειας που θα εξαχθεί στο δίκτυο (exported energy), η ενέργεια που θα ιδιοκαταναλωθεί (self-consumption), οι εκπομπές ρύπων διοξειδίου του άνθρακα (CO2 emissions) όπου υπολογίζονται από τον πολλαπλασιασμό της εισαγόμενης ενέργειας με τον παράγοντα έντασης των ρύπων ανά kWh (χρησιμοποιήθηκε ο μέσος παράγοντας έντασης των ΗΠΑ, περίπου 0.449 kgCO2/kWh) καθώς και η άνεση του χρήστη(comfort) όπου είναι το ποσοστό των κανόνων άνεσης που δίνει ο χρήστης που εκτελούνται. Επίσης μετριέται και ο συνολικός χρόνος εκτέλεσης.

5.2 Αλγόριθμοι

Σε αυτό το υποκεφάλαιο θα αναλυθούν όλοι οι αλγόριθμοι οι οποίοι υλοποιήθηκαν και χρησιμοποιήθηκαν για την πειραματική αξιολόγηση.

5.2.1 Αλγόριθμος Standard

Ο Standard αλγόριθμος παίρνει τα δεδομένα και τα μετατρέπει από ανά λεπτό που είναι όταν διαβάζονται από το αρχείο σε ανά ώρα. Ακολούθως, στα δεδομένα εκτελούνται οι μετρικές έτσι όπως αναλύθηκαν προηγουμένως, και βγαίνουν κάποια στατιστικά πάνω σε αυτές.

Επίσης κατά την διάρκεια του standard αλγόριθμου, ο αλγόριθμος βρίσκει και την μέγιστη κατανάλωση της κάθε συσκευής η οποία θα χρησιμοποιηθεί αργότερα στους υπόλοιπους αλγόριθμους.

Algorithm 1 *Standard*

Input: ECT: Energy Consumption per Minute Table;
Output: ECT^*: Energy Consumption per Hour Table

```
1: Standard(ECT)                                   ▷ Standard Routine
2:   For hour in ECT do                            ▷ hour: subtable with 60 rows (minutes)
3:       applCons ← sum(hour)                      ▷ applCons: appliances consumption per hour
4:       ECT* ← applCons                           ▷ add new consumption
5:   EndFor
6:
7: return (ECT*)
```

Σχήμα 5.1 Ψευδοκώδικας αλγόριθμου Standard

5.2.2 Αλγόριθμος Random

Ο Random αλγόριθμος, είναι αλγόριθμος βελτιστοποίησης. Πιο συγκεκριμένα έχει ως στόχο την μείωση της εισαγωγής ενέργειας από το δίκτυο. Στον αλγόριθμο δίνεται ως είσοδος ο αριθμός των επαναλήψεων των οποίων θα εκτελέσει ο αλγόριθμος.

Αναλυτικότερα ο αλγόριθμος τρέχει ανά μέρα στα δεδομένα και μετατοπίζει τυχαία τις καταναλώσεις των συσκευών μέσα στην μέρα. Έχει την δυνατότητα να στοιβάζει τις

καταναλώσεις μέχρι το σημείο όπου είναι η μέγιστη κατανάλωση της συσκευής όπως υπολογίστηκε προηγουμένως. Αν το αποτέλεσμα της τυχαίας μετατόπισης έχει λιγότερη εισαγωγή ενέργειας τότε αποθηκεύεται, αλλιώς παραβλέπετε. Ο αλγόριθμος θα τρέξει όσες φορές ζητηθεί από τον χρήστη ανάλογα με τον αριθμό που δίνει στην είσοδο.

Algorithm 1 *Random*

Input: ECT: Energy Consumption Table; n: random iterations
Output: ECT^*: New Energy Consumption Table

```
1:  Random(ECT, n)                                          ▷ Random Routine
2:    For day in ECT do                                     ▷ day: sub-table with 24 rows (hours)
3:      i = 0
4:      While i < n do
5:        For hour in day do                                ▷ hour: one row of day
6:          For appl in hour do                             ▷ appl: appliance consumption in current hour
7:            new ← randReallocate(appl)                    ▷ random reallocate appl
8:          EndFor
9:        EndFor
10:       day ← minImp(new, day)                            ▷ compare and keep result with less imported energy
11:       i++
12:     EndWhile
13:     ECT* ← day                                          ▷ save result
14:   EndFor
15:
16: return (ECT*)
```

Σχήμα 5.2 Ψευδοκώδικας αλγόριθμου Random

5.2.3 Αλγόριθμος Brute Force

Ο Brute Force αλγόριθμος έχει ως στόχο την εύρεση της βέλτιστης λύσης, δηλαδή την λύση με την μικρότερη εισαγωγή ενέργειας από το δίκτυο και άρα παράλληλα τις λιγότερες εκπομπές ρύπων διοξειδίου του άνθρακα. Πιο συγκεκριμένα εκτελεί μία κατά βάθος αναζήτηση (Depth-First Search) για να βρει την καλύτερη χρονοδρομολόγηση για την κατανάλωση των συσκευών, με σεβασμό στην μέγιστη κατανάλωση που μπορεί να κάνει μία συσκευή.

Η αναζήτηση εκτελείται ανά μέρα, με στόχο την εύρεση της βέλτιστης χρονοδρομολόγησης της κατανάλωσης ενέργειας των συσκευών, έτσι ώστε να υπάρξει όσων το δυνατόν μεγαλύτερη ιδιοκατανάλωση. Ο αλγόριθμος δεν λαμβάνει καθόλου

υπόψιν το επίπεδο άνεσης του χρήστη, στόχος του είναι η εύρεση της μέγιστης ιδιοκατανάλωσης.

Algorithm 1 $BruteForce$

Input: ECT: Energy Consumption Table;
Output: ECT^*: Optimal Energy Consumption Table

```
1:  BruteForce(ECT)                                  ▷ Brute Force Routine
2:    s = NULL                                       ▷ Stack used for DFS
3:    For day in ECT do                              ▷ day: sub-table with 24 rows (hours)
4:      For hour in day do                           ▷ hour: one row of day
5:        For appl in hour do                        ▷ appl: appliance consumption in current hour
6:          new ← reallocate(appl)                   ▷ reallocate appl
7:          If new < s.peek() Then                   ▷ check imported energy of new and peek of stack
8:            s.push(new)                            ▷ save new in s
9:          Endif
10:       EndFor
11:     EndFor
12:     ECT* ← s.peek()                              ▷ save result
13:   EndFor
14:
15: return (ECT*)
```

Σχήμα 5.3 Ψευδοκώδικας αλγόριθμου Brute Force

5.2.4 Αλγόριθμος GreenCap

Ο αλγόριθμος GreenCap έχει ως βάση τον γενετικό αλγόριθμο. Αναλυτικότερα εκτελούνται κανονικά οι διαδικασίες που έχει ένας γενετικός αλγόριθμος, δηλαδή initialize population, calculate fitness, selection, crossover, mutation, όμως εκτελούνται ακόμα δύο ευρετικές συναρτήσεις τις οποίες υλοποιήσαμε. Η πρώτη συνάρτηση έχει ως στόχο να επαναφέρει την συνολική κατανάλωση εκεί που ήταν στα αρχικά δεδομένα, λόγω τυχών αυξομείωσης που μπορεί να υπάρξει από τις προηγούμενες συναρτήσεις. Η δεύτερη συνάρτηση έχει ως στόχο την εναλλαγή των καταναλώσεων από τις τρεις ώρες αιχμής στις τρεις μη ώρες αιχμής, έτσι όπως υπολογίζονται ανά μέρα στο σύνολο δεδομένων της συνολικής ροής στο ενεργειακό δίκτυο που χρησιμοποιήσαμε. Αν η εναλλαγή έχει λιγότερη εισαγωγή ενέργειας, τότε αποθηκεύεται αλλιώς παραβλέπετε.

Ως fitness function για τον γενετικό αλγόριθμο θέσαμε την συνολική εισαγωγή ενέργειας που γίνεται και ο αλγόριθμος έχει ως στόχο την μείωση της, κάτι που θα μειώσει και την εκπομπή ρύπων διοξειδίου του άνθρακα.

5.2.5 Αλγόριθμος GreenCapComfort

Ο αλγόριθμος GreenCapComfort είναι μία παραλλαγή του αλγόριθμου GreenCap. Οι διαδικασίες και των δύο αλγόριθμων είναι οι ίδιες με την μόνη διαφορά να είναι ότι στον αλγόριθμο GreenCapComfort λαμβάνονται υπόψιν και οι προτιμήσεις του χρήστη, έτσι όπως διαβάζονται μέσα από τους κανόνες που δίνει ο ίδιος ο χρήστης. Κάθε κανόνας ο οποίος εκτελείται έχει μηδενικό κόστος σφάλματος, όμως κάθε κανόνας που δεν εκτελείται χρεώνεται ένα κόστος σφάλματος (υπολογίζεται ανάλογα με το σύνολο των κανόνων).

Ως fitness function του αλγορίθμου χρησιμοποιούνται κάποια βάρη για να αναδειχθεί η σημαντικότητα της εισαγόμενης ενέργειας και του κόστους σφάλματος άνεσης. Πιο συγκεκριμένα η εισαγόμενη ενέργεια λαμβάνεται υπόψιν κατά 75% και το κόστος σφάλματος άνεσης του χρήστη κατά 25%.

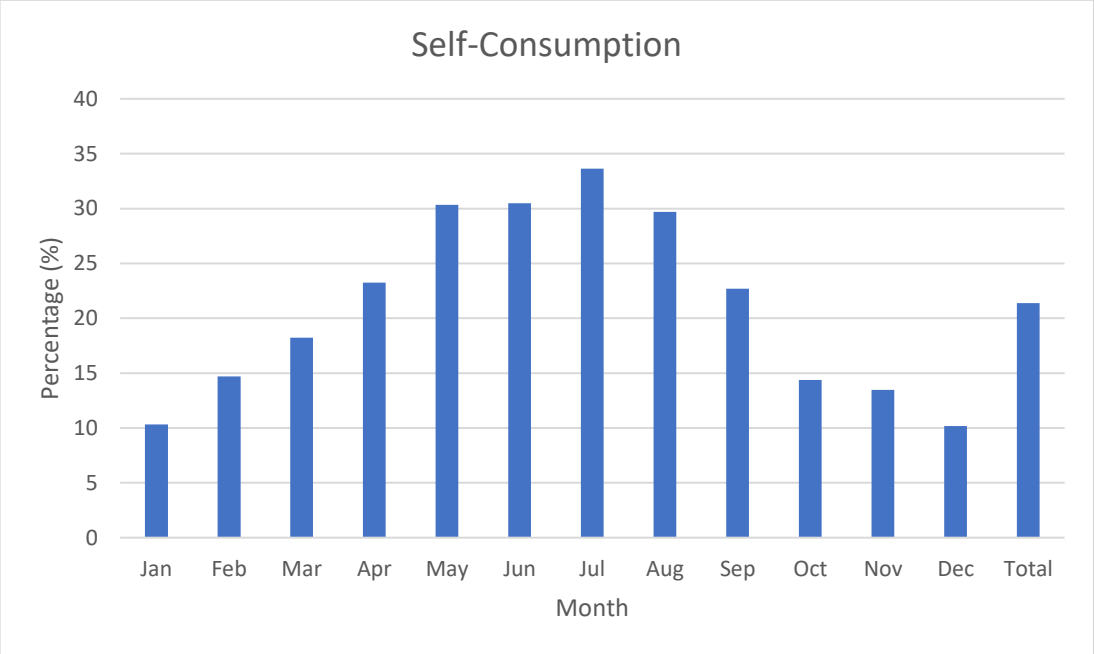
5.3 Αξιολόγηση Απόδοσης

Σε αυτό το υποκεφάλαιο θα αξιολογηθεί η απόδοση του κάθε αλγόριθμου ξεχωριστά και ακολούθως του προτεινόμενου GreenCap framework έναντι των υπόλοιπων αλγορίθμων χρησιμοποιώντας τα ίδια σύνολα δεδομένων.

5.3.1 Αποτελέσματα αλγόριθμου Standard

Ο αλγόριθμος standard αποτελεί μία απλή μετατροπή των δεδομένων από μετρήσεις ανά λεπτό σε ανά ώρα. Μετά την μετατροπή έγιναν κάποιες μετρήσεις χρησιμοποιώντας τις μετρικές που αναφέρθηκαν προηγουμένως και τα αποτελέσματα φαίνονται στα παρακάτω σχήματα.

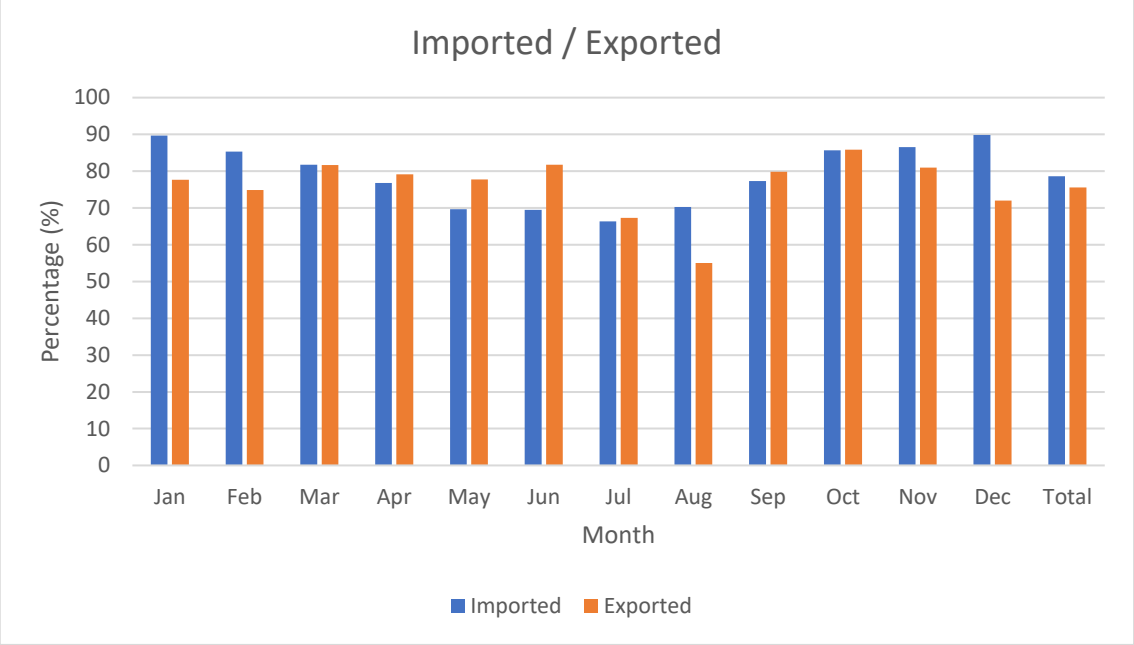
Στο Σχήμα 5.4 όπου είναι τα πραγματικά δεδομένα ενός σπιτιού, βλέπουμε την ιδιοκατανάλωση ηλεκτρικής ενέργειας η οποία γίνεται από τις συσκευές τις οικίας ανά μήνα. Παρατηρούμε ότι το ποσοστό ιδιοκατανάλωσης είναι αρκετά χαμηλό αφού για μήνες όπως ο Γενάρης είναι 10%, ενώ σε κανένα μήνα του χρόνου το ποσοστό ιδιοκατανάλωσης δεν υπερβαίνει το 33% έναντι της κατανάλωσης ενέργειας στον κάθε μήνα. Η συνολική ιδιοκατανάλωση η οποία γίνεται είναι περίπου 20-21% για την διάρκεια του έτους.



Σχήμα 5.4 Ιδιοκατανάλωση αλγορίθμου Standard

Στο Σχήμα 5.5 παρατηρούμε την εισαγόμενη και την εξαγόμενη ενέργεια που γίνεται στην οικία σε σχέση με την κατανάλωση και την παραγωγή ενέργειας.. Παρατηρούμε ότι το ποσοστό της ενέργειας το οποίο εισάγεται από το δίκτυο, με σκοπό την κάλυψη της ανάγκης για ηλεκτρική ενέργεια είναι πάρα πολύ ψηλό. Αναλυτικότερα για τον μήνα Γενάρη, για την κάλυψη της ανάγκης της οικίας για ηλεκτρική ενέργεια πρέπει να εισαχθεί περίπου το 90% της συνολικής κατανάλωσης για τον μήνα αυτό. Παρόμοια και το ποσοστό της ενέργειας το οποίο εξάγεται στο ενεργειακό δίκτυο είναι πολύ ψηλό, άρα παραμένει αχρησιμοποίητο. Για παράδειγμα, τον μήνα Γενάρη το ποσοστό παραγόμενης ενέργειας το οποίο εξάχθηκε στο ενεργειακό δίκτυο είναι περίπου 77%, που σημαίνει ότι το 77% της ενέργειας που παράχθηκε από το φωτοβολταϊκό σύστημα είναι αχρησιμοποίητο.

Από το σχήμα αυτό μπορούμε να παρατηρήσουμε ότι υπάρχει αρκετά μεγάλη προοπτική μέσα από τα δεδομένα που έχουμε, έτσι ώστε η ενέργεια που εξάγεται στο ενεργειακό δίκτυο να ιδιοκαταναλωθεί, μειώνοντας παράλληλα το κόστος της ηλεκτρικής ενέργειας που εισάγεται από το δίκτυο, να μειωθεί το κόστος το οποίο πληρώνεται για την κάλυψη ενέργειας της οικίας, και επιπρόσθετα την μείωση εκπομπών ρύπων διοξειδίου του άνθρακα.



Σχήμα 5.5 Εισαγόμενη / Εξαγόμενη Ενέργεια αλγορίθμου Standard

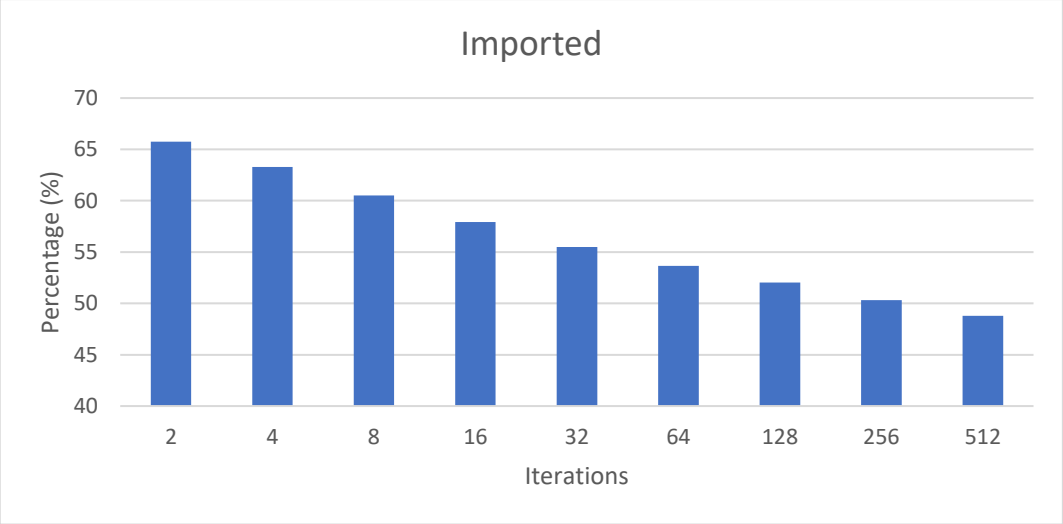
Στον Πίνακα 5.4 φαίνονται αναλυτικά τα συνολικά αποτελέσματα των μετρικών που χρησιμοποιήθηκαν στο σύνολο δεδομένων.

Μετρική	**Αποτέλεσμα**
Imported Energy	~ 7280.88 kWh
Exported Energy	~ 6135.84 kWh
Self-Consumed Energy	~ 1979.22 kWh
Production	~ 8115.08 kWh
Consumption	~ 9260.13 kWh
Comfort	100 %

Πίνακας 5.4 Αποτελέσματα αλγορίθμου Standard

5.3.2 Αποτελέσματα αλγόριθμου Random

Ο αλγόριθμος Random υλοποιήθηκε με στόχο την τυχαία ανακατανομή της κατανάλωσης ενέργειας των συσκευών που βρίσκονται μέσα στην οικία. Στον αλγόριθμο δίνεται ως είσοδος ένας ακέραιος αριθμός όπου διευκρινίζει στην αλγόριθμο πόσες φορές θα γίνει η τυχαία ανακατανομή της κατανάλωσης ανά μέρα που βρίσκεται μέσα στα δεδομένα. Έγιναν αρκετές δοκιμές με βάση τον αριθμό που δίνεται. Όπως μπορούμε να δούμε στο Σχήμα 5.6, όσο μεγαλύτερος είναι ο αριθμός τόσο καλύτερα είναι τα αποτελέσματα τα οποία παίρνουμε, δηλαδή έχουμε λιγότερη εισαγόμενη ενέργεια. Όμως όσο μεγαλύτερος είναι ο αριθμός ο οποίος δίνεται ως είσοδος τόσο μεγαλύτερος είναι και ο χρόνος εκτέλεσης του αλγορίθμου έτσι όπως φαίνεται στο Σχήμα 5.6 .

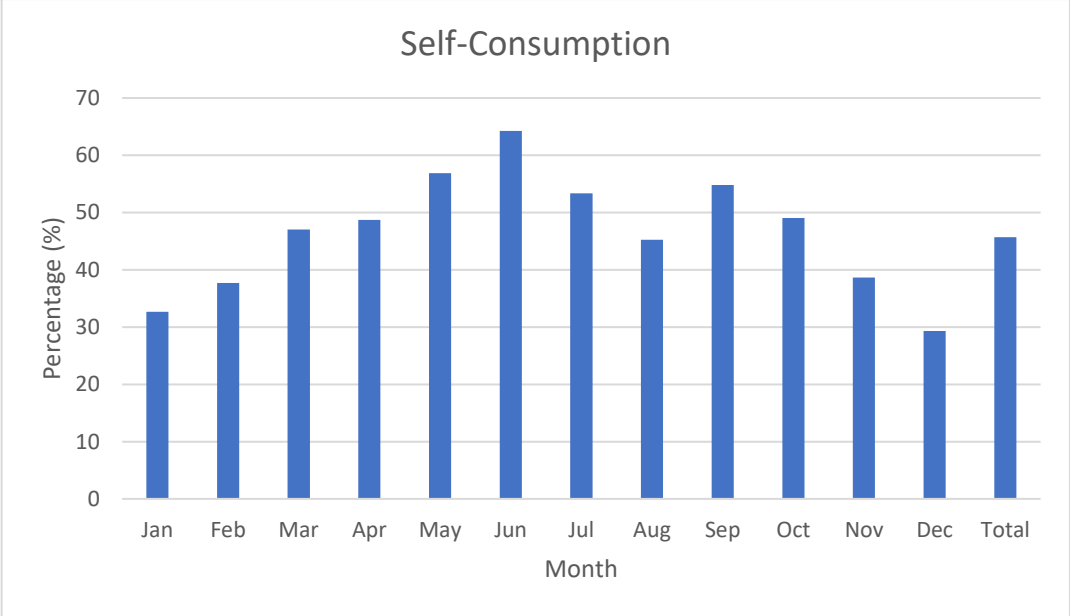


Σχήμα 5.6 Συνολική Εισαγόμενη Ενέργεια αλγορίθμου Random

Σχήμα 5.7 Χρόνος Εκτέλεσης αλγορίθμου Random

Ο λόγος που τα αποτελέσματα είναι καλύτερα για τις προσπάθειες με πιο πολλές επαναλήψεις, είναι διότι σε κάθε επανάληψη γίνεται αξιολόγηση της λύσης που επιστρέφει ο Random αλγόριθμος με βάση την εισαγόμενη ενέργεια και αν είναι λιγότερη τότε η λύση αποθηκεύεται, αλλιώς παραβλέπετέ.

Όπως και στον αλγόριθμο Standard, στο σχήμα 5.8 φαίνεται η ιδιοκατανάλωση που γίνεται κάθε μήνα, καθώς και η μέση και η συνολική ιδιοκατανάλωση για ένα παράδειγμα εκτέλεσης του αλγορίθμου Random με 50 επαναλήψεις.



Σχήμα 5.8 Ιδιοκατανάλωση αλγορίθμου Random

Μετρική	**Αποτέλεσμα**
Επαναλήψεις	50
Imported Energy	~ 5520.91 kWh
Exported Energy	~ 4375.87 kWh
Self-Consumed Energy	~ 3739.20 kWh
Production	~ 8115.06 kWh
Consumption	~ 9260.13 kWh
Comfort	~ 20%

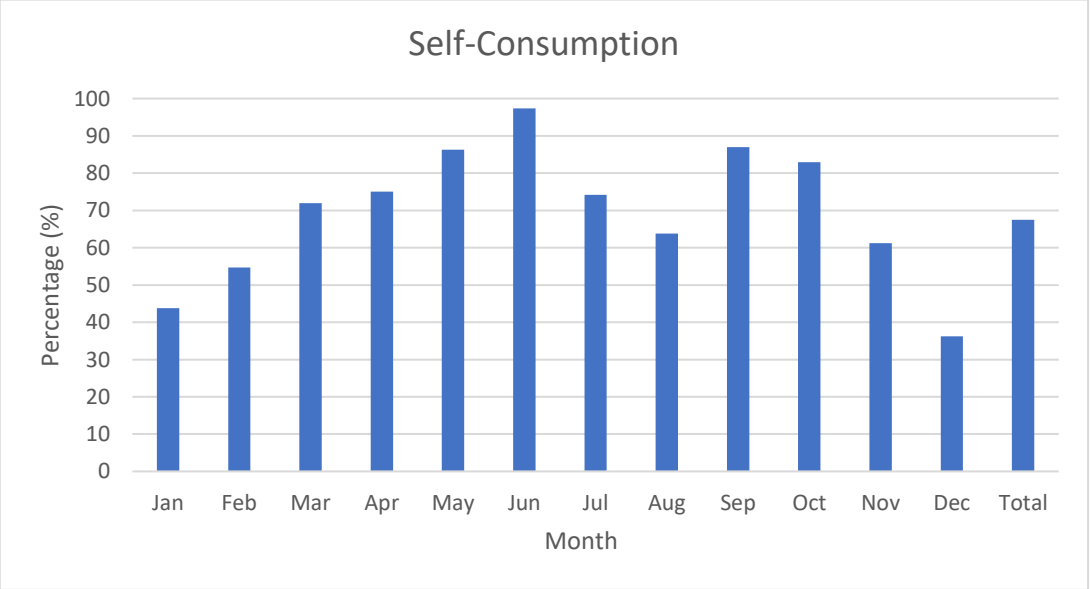
Πίνακας 5.5 Αποτελέσματα αλγορίθμου Random

Στον Πίνακα 5.5 φαίνονται αναλυτικά τα αποτελέσματα για ένα παράδειγμα εκτέλεσης του αλγόριθμου Random με 50 επαναλήψεις.

5.3.3 Αποτελέσματα αλγόριθμου Brute Force

Ο αλγόριθμος του Brute Force υλοποιήθηκε για να μας δώσει ένα βέλτιστο αποτέλεσμα, έτσι ώστε να γίνει η σύγκριση των υπόλοιπων αλγορίθμων με αυτόν. Ο αλγόριθμος χρησιμοποιεί ένα πιο light version του Depth First Search για να βρει την βέλτιστη χρονοδρομολόγηση για την κατανάλωση ενέργειας από τις συσκευές, ανά μέρα. Πιο συγκεκριμένα εναλλάσσει τις καταναλώσεις ενέργειας που κάνουν οι συσκευές μέσα σε μια μέρα μέχρι να βρεθεί η βέλτιστη χρονοδρομολόγηση. Υπάρχουν κάποια περιθώρια βελτίωσης του αλγορίθμου, καθώς δεν υλοποιήθηκε ο τεμαχισμός της κατανάλωσης ενέργειας από μία συσκευή σε δύο ή περισσότερες ώρες.

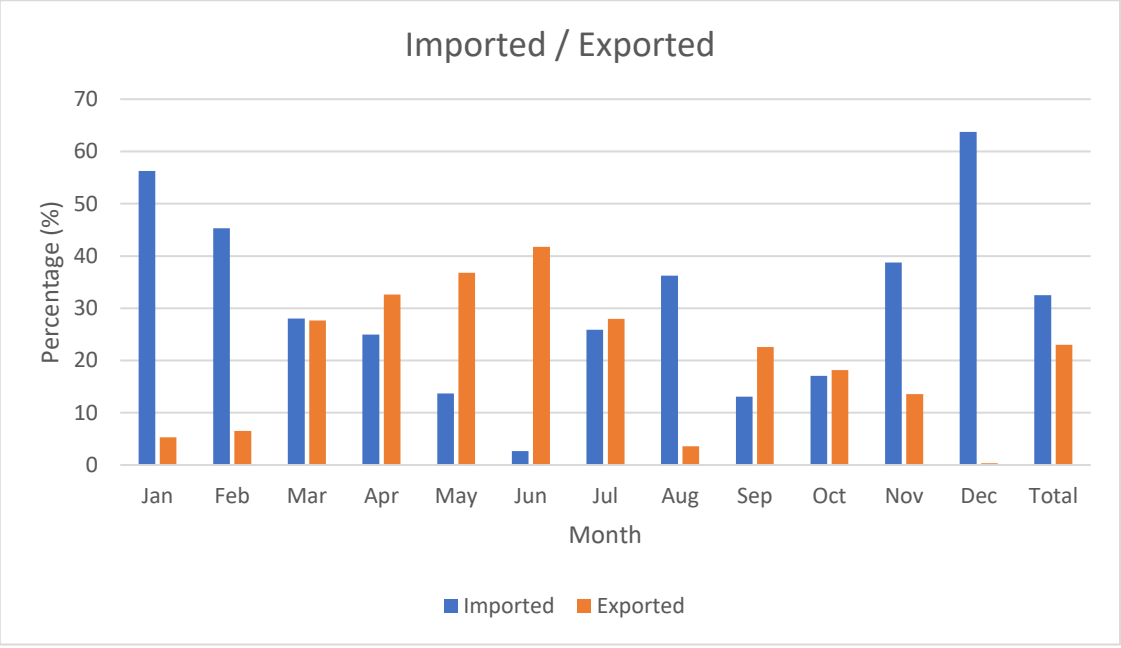
Έγιναν αρκετές δοκιμές του αλγορίθμου και τα αποτελέσματα τα οποία πήραμε είναι τα ακόλουθα. Αρχικά στο Σχήμα 5.9 μπορούμε να δούμε την ιδιοκατανάλωση που γίνεται ανά μήνα, καθώς και την συνολική ιδιοκατανάλωση. Παρατηρούμε σε αυτήν την περίπτωση ότι το ποσοστό ιδιοκατανάλωσης σε όλους του μήνες είναι αρκετά ψηλό. Για παράδειγμα, τον μήνα Ιούνιο η ιδιοκατανάλωση είναι σχεδόν 98%, κάτι ιδιαίτερα αξιοσημείωτο.



Σχήμα 5.9 Ιδιοκατανάλωση αλγορίθμου Brute Force

Επίσης στο Σχήμα 5.10 παρατηρούμε ότι σε σχέση με το Σχήμα 5.5 όπου παρουσιάζεται το ποσοστό της ενέργειας που εισάγεται από το ενεργειακό δίκτυο σε σχέση με την κατανάλωση ενέργειας που γίνεται καθώς επίσης και το ποσοστό ενέργειας που εξάγεται στο δίκτυο σε σχέση με την παραγωγή ενέργειας από το φωτοβολταϊκό σύστημα, ότι υπάρχει αρκετά μεγάλη διαφορά σε σχέση με τα αρχικά δεδομένα. Πιο συγκεκριμένα, για τον μήνα Μάρτιο, μετά τον αλγόριθμο Brute Force, η εισαγωγή και εξαγωγή ενέργειας από και προς το ενεργειακό δίκτυο είναι περίπου 28%. Στα αρχικά μας δεδομένα τον μήνα Μάρτη η εισαγωγή και εξαγωγή ενέργειας ήταν περίπου 81%.

Αυτό μας φανερώνει πόσο περιθώριο υπάρχει έτσι ώστε να βελτιωθεί η χρονοδρομολόγηση στην κατανάλωση ενέργειας από τις συσκευές στην οικία, με σκοπό την μείωση της εισαγωγής ενέργειας από το ενεργειακό δίκτυο, μειώνοντας παράλληλα και την εκπομπή ρύπων διοξειδίου του άνθρακα.



Σχήμα 5.10 Εισαγόμενη / Εξαγόμενη Ενέργεια αλγορίθμου Brute Force

Στον Πίνακα 5.6 φαίνονται αναλυτικά τα αποτελέσματα για ένα παράδειγμα εκτέλεσης του αλγόριθμου Brute Force.

Μετρική	**Αποτέλεσμα**
Imported Energy	~ 3011.32 kWh
Exported Energy	~ 1866.29 kWh
Self-Consumed Energy	~ 6248.77 kWh
Production	~ 8115.06 kWh
Consumption	~ 9260.13 kWh
Comfort	~ 10%

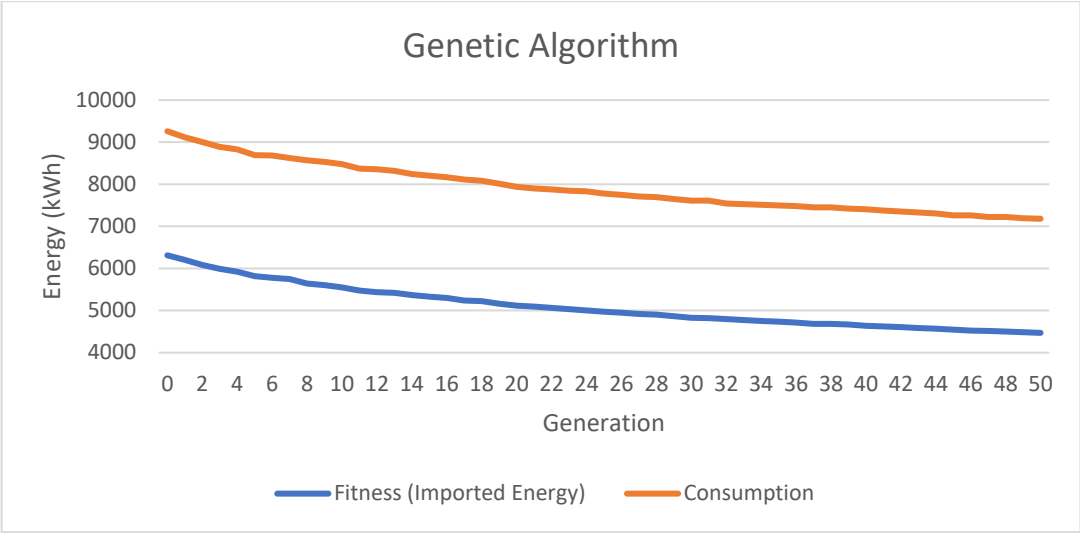
Πίνακας 5.6 Αποτελέσματα αλγορίθμου Brute Force

5.3.4 Αποτελέσματα αλγορίθμου GreenCap

Ο αλγόριθμος GreenCap υλοποιήθηκε με βάση τον γενετικό αλγόριθμος. Στον αλγόριθμος υπάρχουν διάφοροι παράμετροι οι οποίοι μπορούν να αλλάξουν, όπως για παράδειγμα ο αριθμός των γενιών, ο πληθυσμός λύσεων ο οποίος δημιουργείται από τον αλγόριθμο, το fitness function του αλγορίθμου, δηλαδή σε ποια παράμετρο γίνεται η αξιολόγηση της κάθε λύσης. Θα γίνει πλήρης ανάλυση όλων των παραμέτρων του αλγορίθμου με παραδείγματα ακολούθως.

Αρχικά ο αλγόριθμος GreenCap αποτελείται από 7 υπορουτίνες. Οι πρώτες 5 υπορουτίνες αποτελούν τα πέντε στάδια ενός γενετικού αλγορίθμου. Η επόμενη υπορουτίνα είναι η διόρθωση της κατανάλωσης ενέργειας που γίνεται από τις συσκευές, και η τελευταία υπορουτίνα είναι η ανακατανομή από τις ώρες αιχμής στις μη ώρες αιχμής. Κατά την διάρκεια υλοποίησης του αλγορίθμου, ξεκινήσαμε με την υλοποίηση του γενετικού αλγόριθμου, δηλαδή των 5 υπορουτίνων του initial population, selection, crossover, mutation και υπολογισμός του fitness.

Ο γενετικός αλγόριθμος από μόνος του δεν μπορούσε να μας δώσει τα επιθυμητά αποτελέσματα. Όντως ο γενετικός αλγόριθμος κατάφερνε να μειώσει την εισαγωγή ενέργειας από το ενεργειακό δίκτυο, όμως παράλληλα με την μείωση της εισαγόμενης ενέργειας από το ενεργειακό δίκτυο είχαμε και την μείωση της συνολικής κατανάλωσης που γινόταν στην οικία, κάτι που είναι ανεπιθύμητο καθώς δεν θα μπορούσε να καλύψει τις ενεργειακές ανάγκες της οικίας.



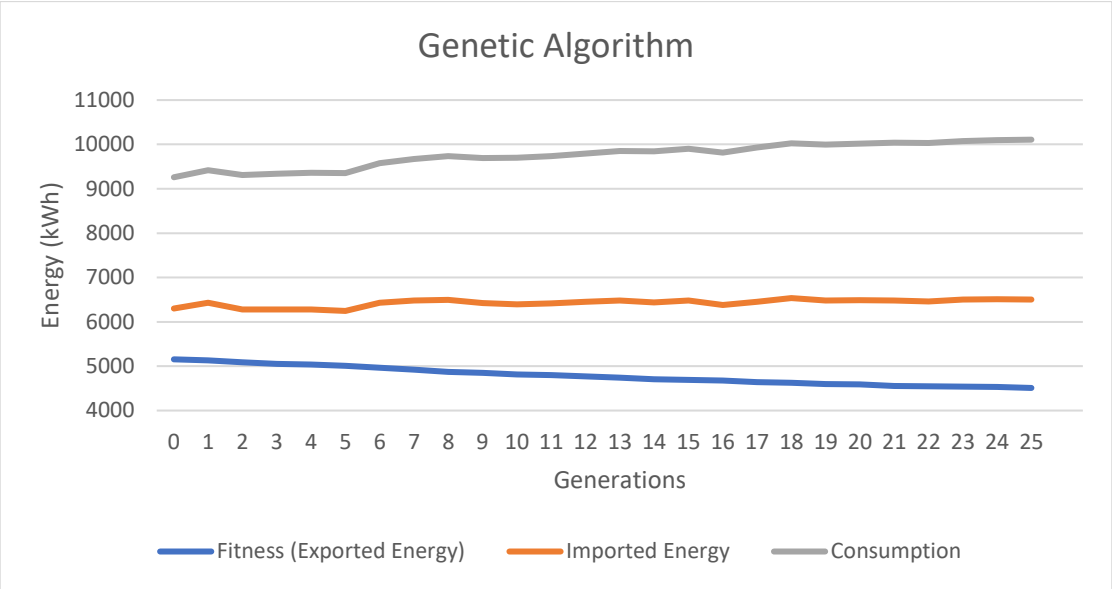
Σχήμα 5.11 Αποτελέσματα Γενετικού Αλγορίθμου (Fitness: Imported)

Στο Σχήμα 5.11 φαίνεται το πρώτο παράδειγμα εκτέλεσης του γενετικού αλγορίθμου χρησιμοποιώντας ως fitness function την συνολική εισαγόμενη ενέργεια από το ενεργειακό δίκτυο, όπου φαίνεται ξεκάθαρα αυτό που προαναφέρθηκε. Ο γενετικός αλγόριθμος, αντί να μετατοπίσει την κατανάλωση ενέργειας στις ώρες όπου υπάρχει κατανάλωση όπως φαίνεται στον Πίνακα 5.7, αυτό που κάνει είναι να μειώσει την συνολική κατανάλωση ενέργειας που γίνεται, και αυτό τον τρόπο μειώνετε και η εισαγωγή ενέργειας που γίνεται από το ενεργειακό δίκτυο. Στον Πίνακα 5.7 φαίνεται πριν και μετά την συνάρτηση crossover. Φαίνεται ότι η συνάρτηση crossover μειώνει την εισαγόμενη ενέργεια αλλά μειώνει ταυτόχρονα και την συνολική κατανάλωση της συσκευής.

Offspring 1 Before			Offspring 2 Before		
Hour	Consumption	Production	Hour	Consumption	Production
0:00	0	0	0:00	3	0
4:00	2	0	4:00	4	0
8:00	4	1	8:00	0	1
12:00	1	5	12:00	1	5
16:00	0	3	16:00	2	3
20:00	3	0	20:00	0	0

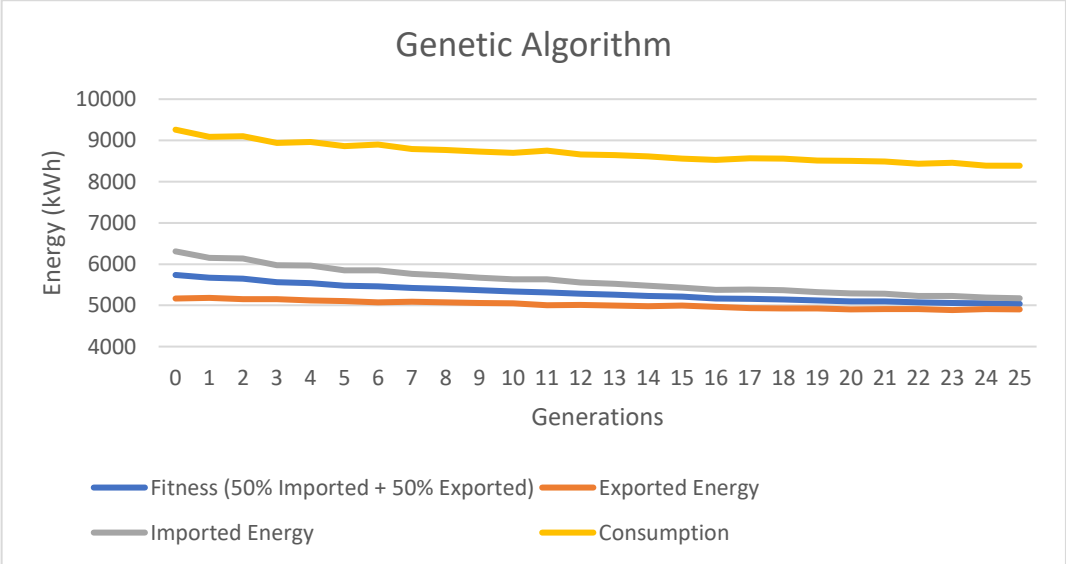
Offspring 1 After			Offspring 2 After		
Hour	Consumption	Production	Hour	Consumption	Production
0:00	3	0	0:00	0	0
4:00	4	0	4:00	2	0
8:00	4	1	8:00	0	1
12:00	1	5	12:00	1	5
16:00	0	3	16:00	2	3
20:00	3	0	20:00	0	0

Πίνακας 5.7 Πριν και μετά το Crossover



Σχήμα 5.12 Αποτελέσματα Γενετικού Αλγορίθμου (Fitness: Exported)

Στο παράδειγμα εκτέλεσης στο Σχήμα 5.12 χρησιμοποιήθηκε ως fitness function η εξαγόμενη ενέργεια. Σε αυτή την περίπτωση παρατηρούμε ότι η κατανάλωση ενέργειας αυξάνεται αφού για την μείωση της εξαγωγής ενέργειας, ο γενετικός αλγόριθμος αυξάνει την ενέργεια που καταναλώνεται στις ώρες όπου υπάρχει παραγωγή. Όμως παρατηρούμε επίσης ότι η εισαγόμενη ενέργεια δεν μειώνεται, αντιθέτως μένει στα ίδια επίπεδα, που αυτό δεν είναι επιθυμητό καθώς δεν μειώνονται οι εκπομπές ρύπων διοξειδίου του άνθρακα που έχουμε θέσει ως στόχο.



Σχήμα 5.13 Αποτελέσματα Γενετικού Αλγορίθμου (Fitness: 50% Exported + 50% Imported)

Στο παράδειγμα εκτέλεσης στο Σχήμα 5.13 χρησιμοποιήθηκε ως fitness function και η εισαγόμενη(50%) και η εξαγόμενη(50%) ενέργεια. Και σε αυτήν την περίπτωση παρατηρούμε ότι επιτυγχάνουμε την μείωση της εισαγωγής ενέργειας, όπως και στο παράδειγμα στο Σχήμα 5.11 όμως έχουμε παράλληλα και μείωση της κατανάλωσης ενέργειας κάτι που δεν είναι επιθυμητό. Σε αυτήν την περίπτωση όμως σε αντίθεση με το Σχήμα 5.12 η μείωση της συνολικής κατανάλωσης ενέργειας δεν είναι τόσο μεγάλη.

Στον Πίνακα 5.7 φαίνονται αναλυτικά τα αποτελέσματα για ένα παράδειγμα εκτέλεσης του Γενετικού Αλγόριθμου.

Μετρική	**Αποτέλεσμα**
Fitness Function	50% Imported + 50% Exported
Population Size	50
Generations	50
Imported Energy	~ 4448.39 kWh
Exported Energy	~ 5322.16 kWh
Self-Consumed Energy	~ 2792.90 kWh
Production	~ 8115.06 kWh
Consumption	~ 7241.33 kWh
Comfort	~ 40%

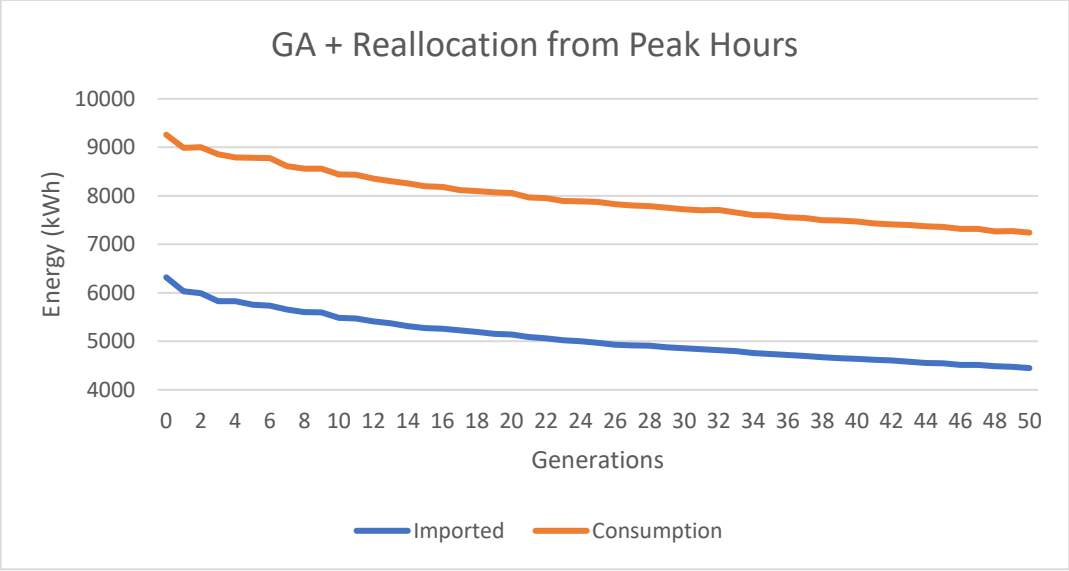
Πίνακας 5.8 Αποτελέσματα Γενετικού Αλγόριθμου

Μετά από τα προηγούμενα παραδείγματα συμπεραίναμε ότι μόνος του ο γενετικός αλγόριθμος δεν μπορεί να λύσει το πρόβλημα της χρονοδρομολόγησης των συσκευών με στόχο την μείωση της εισαγόμενης ενέργειας. Ακολούθως υλοποιήσαμε την ευρετική συνάρτηση για ανακατανομή από τις ώρες αιχμής στις μη ώρες αιχμής.

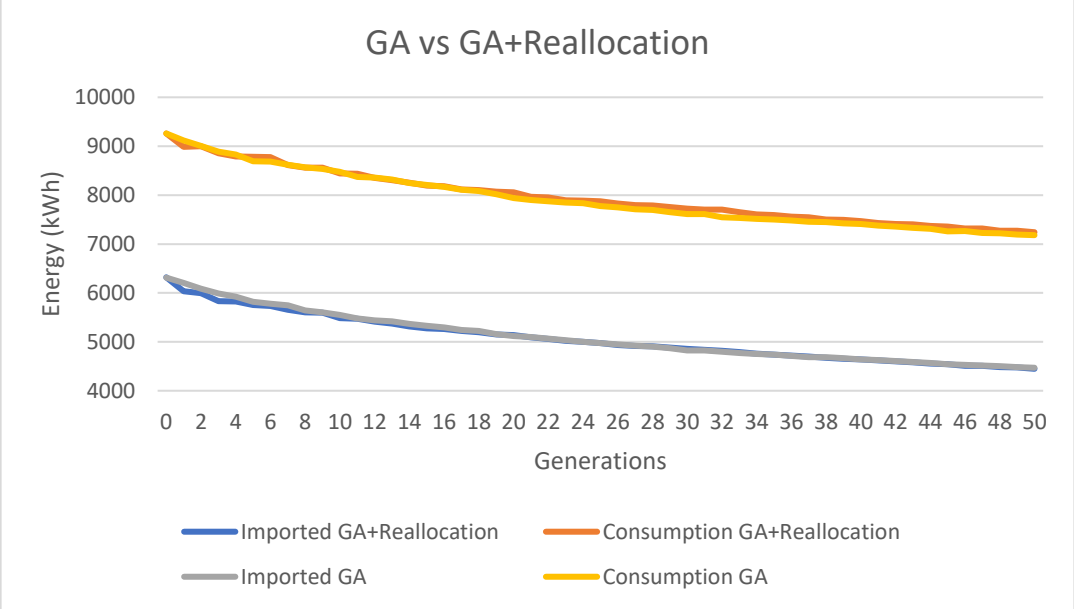
Η ευρετική συνάρτηση της ανακατανομής της κατανάλωσης ενέργειας από τις ώρες αιχμή στις μη ώρες αιχμής, δεν βοηθά σε πολύ μεγάλο βαθμό την μείωση της εισαγόμενης ενέργειας στο σπίτι. Ο λόγος που υλοποιήθηκε η συνάρτηση αυτή είναι για την εκμετάλλευση του Demand Response, δηλαδή την μετατόπιση της κατανάλωσης ενέργειας από τις ώρες αιχμής όπου το κόστος της ηλεκτρικής ενέργειας που εισάγεται από το ενεργειακό δίκτυο είναι υψηλότερο σε σύγκριση με τις μη ώρες αιχμής όπου το

κόστος της εισαγόμενης ενέργειας από το ενεργειακό δίκτυο είναι αρκετά χαμηλότερο. Αναλυτικότερα μετά την υλοποίηση της ευρετικής συνάρτησης ανακατανομής ενέργειας από τις ώρες αιχμής στις μη ώρες αιχμής έχουμε τα ακόλουθα αποτελέσματα.

Στο Σχήμα 5.14 όπως και προηγουμένως στο παράδειγμα εκτέλεσης του γενετικού αλγορίθμου (χρησιμοποιώντας ως fitness function το imported energy – Σχήμα 5.11) παρατηρούμε ότι η συνολική κατανάλωση μειώνεται, κάτι που είναι ανεπιθύμητο.



Σχήμα 5.14 Αποτελέσματα Γενετικού Αλγορίθμου + Ανακατανομή (Fitness: Imported)



Σχήμα 5.15 Σύγκριση Χρήσης και μη Χρήσης Ανακατανομής στον Γενετικό Αλγόριθμο

Στο Σχήμα 5.15, παρατηρούμε ότι τα αποτελέσματα του γενετικού με την χρήση της ευρετικής συνάρτησης ανακατανομής από τις ώρες αιχμής στις μη ώρες αιχμής και χωρίς την χρήση της είναι πολύ παρόμοια. Όμως υπάρχει μία ουσιώδης διαφορά ανάμεσα τους, και αυτή είναι το συνολικό κόστος της ενέργειας που εισάγεται από το ενεργειακό δίκτυο. Το συνολικό κόστος της ενέργειας που εισάγεται από το ενεργειακό δίκτυο στην περίπτωση που γίνεται χρήση της συνάρτησης ανακατανομής από τις ώρες αιχμής στις μη ώρες αιχμής είναι μικρότερο, επειδή εκμεταλλεύεται το Demand Response, δηλαδή ότι στις μη ώρες αιχμής η ενέργεια είναι φθηνότερη.

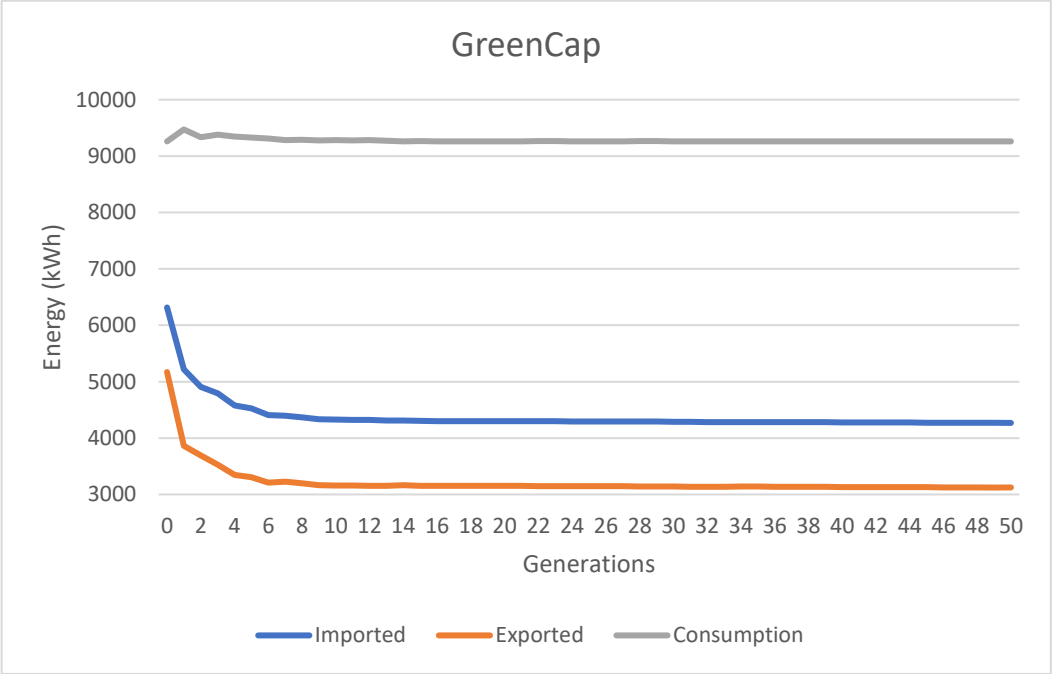
Τα αναλυτικά αποτελέσματα του γενετικού αλγορίθμου με την χρήση της ευρετικής συνάρτησης ανακατανομής φαίνονται στον Πίνακα 5.8 .

Μετρική	**Αποτέλεσμα**
Reallocation	Yes
Fitness Function	Imported
Population Size	50
Generations	50
Imported Energy	~ 4658.31 kWh
Exported Energy	~ 4903.12 kWh
Self-Consumed Energy	~ 3211.94 kWh
Production	~ 8115.06 kWh
Consumption	~ 7870.26 kWh
Comfort	~ 20%

Πίνακας 5.9 Αποτελέσματα Γενετικού Αλγόριθμου + Ανακατανομή (Fitness: Imported)

Μετά από πάρα πολλές προσπάθειες η οποίες έγιναν, με στόχο να μην μειώνεται η συνολική κατανάλωση ενέργειας, αποφασίστηκε η υλοποίηση μίας επιπλέον ευρετικής συνάρτησης, όπου θα έχει ως στόχο την επαναφορά της κατανάλωσης ενέργειας στα επίπεδα πριν να γίνει οποιαδήποτε διαδικασία πάνω στα δεδομένα, δηλαδή στα επίπεδα όπου η κατανάλωση ενέργειας βρίσκεται μετά την εκτέλεση του αλγορίθμου Standard. Για να γίνει αυτό έπρεπε πριν την εκτέλεση οποιασδήποτε συνάρτησης να βρεθεί η συνολική κατανάλωση ενέργειας που γίνεται από τις συσκευές για κάθε μέρα που υπάρχει στα δεδομένα μας.

Έχοντας βρει πόση είναι η συνολική κατανάλωση των συσκευών κάθε μέρα, ακολούθως πριν την εκτέλεση της συνάρτησης ανακατανομής εκτελείται η συνάρτηση διόρθωσης της συνολικής κατανάλωσης των οικιακών συσκευών. Για την εκμετάλλευση αυτής της διόρθωσης η οποία γίνεται, επιχειρούμε η διόρθωση αυτή να γίνει με άπληστο τρόπο. Πιο συγκεκριμένα, αν σε μία συσκευή αυξηθεί η συνολική κατανάλωση η οποία γίνεται κατά την διάρκεια μίας μέρας, τότε η μείωση η οποία γίνεται είναι στις ώρες που δεν υπάρχει παραγωγή ενέργειας, και έτσι μειώνεται και η εισαγόμενη ενέργεια από το ενεργειακό δίκτυο. Αντίθετα αν σε μία συσκευή μειωθεί η συνολική κατανάλωση κατά την διάρκεια μιας μέρα, η αύξηση κατανάλωσης ενέργειας γίνεται στις ώρες όπου υπάρχει παραγωγή ενέργειας, και με αυτό τον τρόπο αποφεύγεται η αύξηση της εισαγωγής ενέργειας και παράλληλα μειώνεται η εξαγωγή ενέργειας προς το ενεργειακό δίκτυο.



Σχήμα 5.16 Αποτελέσματα αλγορίθμου GreenCap (Fitness: Imported)

Στο Σχήμα 5.16 παρατηρούμε αυτό ακριβώς που προαναφέρθηκε, η συνολική κατανάλωση ενέργειας παραμένει ανεπηρέαστη (παρά τα αρχικά σκαμπανεβάσματα) και η εισαγωγή και εξαγωγή ενέργειας από και προς το ενεργειακό δίκτυο να μειώνεται, κάτι που είναι επιθυμητό γιατί οι δύο αυτές μετρικές εξαρτώνται η μία από την άλλη αν η συνολική κατανάλωση ενέργειας παραμένει ανεπηρέαστη. Επίσης με την χρήση και της

συνάρτησης ανακατανομής από τις ώρες αιχμής στις μη ώρες αιχμής, επιτυγχάνουμε να μειώσουμε και το κόστος της ενέργειας που εισάγεται από το ενεργειακό δίκτυο.

Τα αναλυτικά αποτελέσματα του αλγόριθμου GreenCap φαίνονται στον Πίνακα 5.9 .

Μετρική	**Αποτέλεσμα**
Reallocation	Yes
Fix Consumption	Yes
Fitness Function	Imported
Population Size	50
Generations	50
Imported Energy	~ 4268.55 kWh
Exported Energy	~ 3123.27 kWh
Self-Consumed Energy	~ 4991.79 kWh
Production	~ 8115.06 kWh
Consumption	~ 9260.33 kWh
Comfort	~ 30%

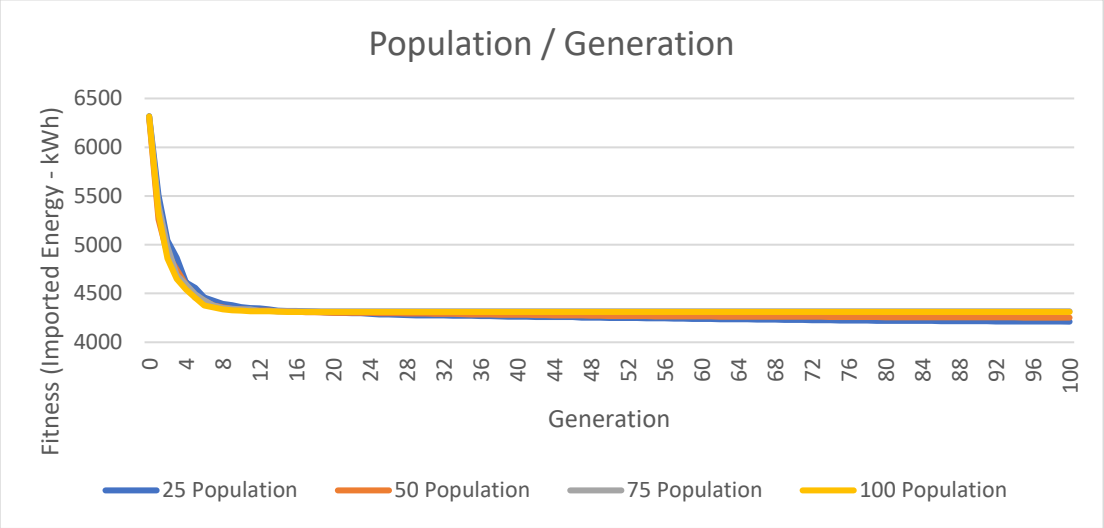
Πίνακας 5.10 Αποτελέσματα αλγορίθμου GreenCap

Στην συνέχεια και στην προσπάθεια εύρεσης των βέλτιστων παραμέτρων για τον αλγόριθμο έγιναν αρκετές δοκιμές σχετικά με τον αριθμό των γενιών που τρέχει ο αλγόριθμος GreenCap και βγήκαν τα ακόλουθα συμπεράσματα. Αρχικά είναι κατανοητό ότι όσο περισσότερες γενιές τρέχουν, τόσο μεγαλύτερος είναι ο χρόνος εκτέλεσης του αλγορίθμου. Όπως φαίνεται από το Σχήμα 5.17, περισσότερες γενιές δεν σημαίνει απαραίτητα και καλύτερο αποτέλεσμα. Πιο συγκεκριμένα φαίνεται μέσα από το σχήμα αυτό ότι στις πρώτες περίπου 10 γενιές η μείωση της εισαγόμενης ενέργειας από το ενεργειακό δίκτυο είναι αρκετά μεγάλη. Στις επόμενες γενιές, παρατηρούμε ότι η μείωση αυτή γίνεται όλο και μικρότερη με αποτέλεσμα να συμπεραίνουμε ότι δεν χρειάζεται να τρέξει ο αλγόριθμος παραπάνω από 10 γενιές.

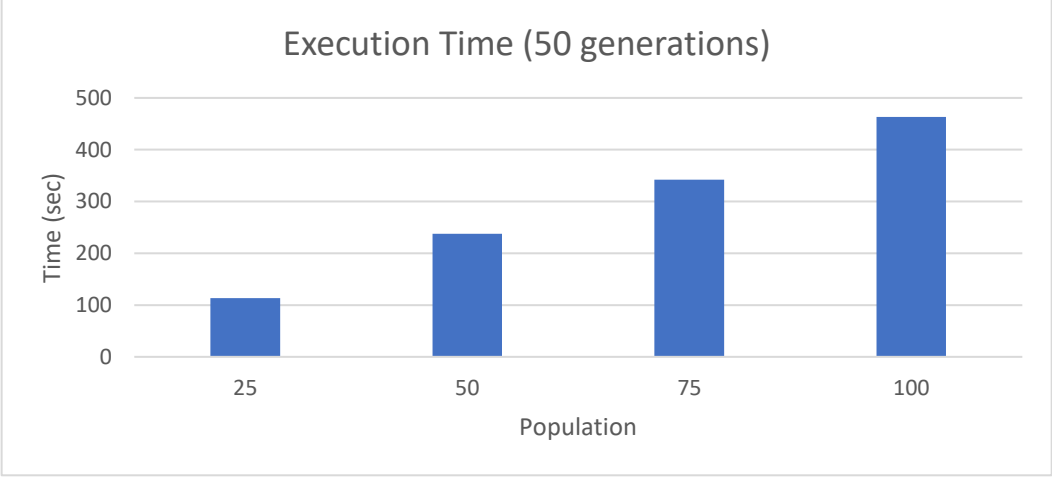
Επίσης κατά την διάρκεια των πειραμάτων έγιναν δοκιμές και για διαφορετικό αριθμό από πληθυσμούς όπως φαίνεται ξανά στο Σχήμα 5.17 . Παρόμοια με τις γενιές έτσι και στην επιλογή πληθυσμού όσο μεγαλύτερος είναι ο πληθυσμός του γενετικού αλγόριθμου

ο οποίος επιλέγουμε, τόσο περισσότερος είναι και ο χρόνος εκτέλεσης του αλγορίθμου όπως φαίνεται στο Σχήμα 5.18 .

Λόγω του randomness το οποίο υπάρχει στον γενετικό αλγόριθμο επιλέχτηκε όπως να χρησιμοποιούμε πληθυσμό ίσο με 50 επειδή μας δίνει αρκετά μεγάλο εύρος από τυχαίες λύσεις, και έτσι είναι ευκολότερο από τον αλγόριθμο να βρει πιο εύκολα την βέλτιστη χρονοδρομολόγηση. Επίσης οι γενιές οι οποίες θα τρέξουν για κάθε δοκιμή και εκτέλεση του αλγορίθμου GreenCap είναι ίσες με 10 για τους λόγους οι οποίοι εξηγήθηκαν προηγουμένως, δηλαδή η οποιαδήποτε μείωση στο fitness του αλγορίθμου σημειώνεται κατά τις πρώτες 10 γενιές. Περισσότερες γενιές δεν μας δίνουν ιδιαίτερα καλύτερο αποτέλεσμα.



Σχήμα 5.17 Αποτελέσματα GreenCap ανά γενιά και πληθυσμό



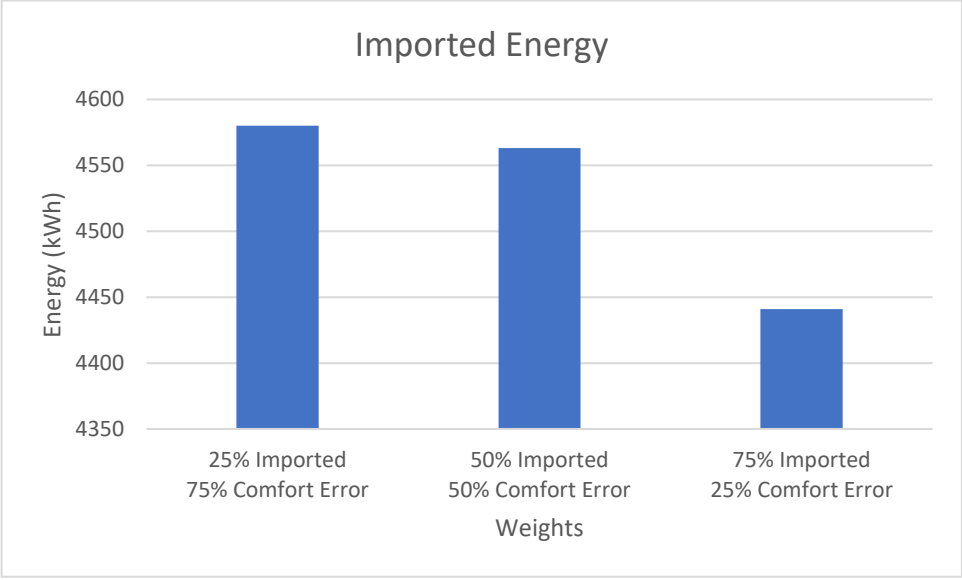
Σχήμα 5.18 Σύγκριση χρόνου εκτέλεσης για διάφορες γενιές

5.3.5 Αποτελέσματα αλγορίθμου GreenCapComfort

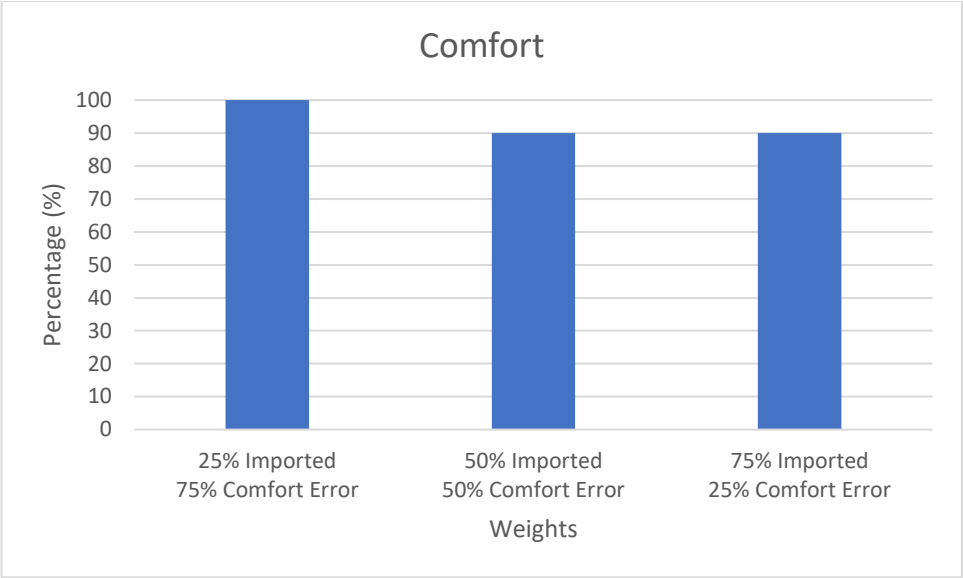
Ακολούθως και αφού παρατηρήσαμε ότι, ενώ ο αλγόριθμος GreenCap μας δίνει ικανοποιητικά αποτελέσματα ως προς την μείωση της εισαγόμενης ενέργειας, και παράλληλα την μείωση των εκπομπών ρύπων διοξειδίου του άνθρακα (CO2), τα αποτελέσματα ως προς το επίπεδο άνεσης του χρήστη δεν ήταν ικανοποιητικά καθώς μόνο το περίπου 30% των κανόνων άνεσης που δίνει ο χρήστης ικανοποιούνται.

Για την αντιμετώπιση αυτής της κατάστασης υλοποιήσαμε ακόμα μία έκδοση του αλγορίθμου GreenCap και την ονομάσαμε GreenCapComfort. Η έκδοση αυτή του αλγορίθμου, έχει ακριβώς τις ίδιες διαδικασίες με την κανονική έκδοση του GreenCap όμως σε αυτήν την περίπτωση στο fitness function λαμβάνουμε υπόψιν και την άνεση του χρήστη. Αναλυτικότερα, όταν ο αλγόριθμος αξιολογεί τα αποτελέσματα που επιστρέφονται σε κάθε νέα λύση, χρεώνεται κάποιο κόστος σφάλματος άνεσης σε σχέση με τους κανόνες άνεσης του χρήστη που δεν εκτελούνται. Το fitness function λαμβάνει υπόψιν και την εισαγόμενη ενέργεια και το κόστος σφάλματος άνεσης του χρήστη, χρησιμοποιώντας κάποιο βάρος σε αυτά τα δύο, έτσι ώστε να μειώνεται και η εισαγόμενη ενέργεια από το ενεργειακό δίκτυο και το κόστος σφάλματος την άνεσης του χρήστη.

Για την εύρεση του ιδανικού βάρους σε κάθε περίπτωση εκτελέσαμε διάφορα παραδείγματα, όπως φαίνονται στην συνέχεια.



Σχήμα 5.19 Σύγκριση βαρών με εισαγόμενη ενέργεια στον αλγόριθμο GreenCapComfort



Σχήμα 5.20 Σύγκριση βαρών με άνεση του χρήστη στον αλγόριθμο GreenCapComfort

Από το Σχήμα 5.19 και Σχήμα 5.20 φαίνεται χαρακτηριστικά η διαφορά η οποία υπάρχει στα αποτελέσματα του αλγόριθμου όταν χρησιμοποιούνται διάφορα βάρη στην fitness function του αλγορίθμου GreenCapComfort. Αποφασίσαμε ότι τα καλύτερα αποτελέσματα μας τα δίνει όταν χρησιμοποιούμε ως βάρη το 75% της εισαγόμενης ενέργειας από το ενεργειακό δίκτυο και το 25% της κόστος σφάλματος άνεσης του χρήστη. Ο λόγος της επιλογής αυτής είναι γιατί, αρχικά ο αλγόριθμος σχεδιάστηκε με στόχο την μείωση της εισαγόμενης ενέργειας από το ενεργειακό δίκτυο και παράλληλα την μείωση των εκπομπών ρύπων CO2. Όσο περισσότερο λαμβάνουμε υπόψιν την μεταβλητή αυτή στον αλγόριθμος τόσο πιο πολύ μειώνεται. Επίσης, ο αλγόριθμος επιτυγχάνει να κρατά αρκετά ψηλά και την άνεση του χρήστη, καθώς όπως μπορούμε να δούμε η άνεση του χρήστη παραμένει περίπου στο 90%.

Τα αναλυτικά αποτελέσματα του αλγόριθμου GreenCap φαίνονται στον Πίνακα 5.10 .

Μετρική	Αποτέλεσμα
Reallocation	Yes
Fix Consumption	Yes
Fitness Function	75% Imported + 25% Comfort Error
Population Size	30
Generations	10
Imported Energy	~ 4420.40 kWh
Exported Energy	~ 3268.06 kWh
Self-Consumed Energy	~ 4847.00 kWh
Production	~ 8115.06 kWh
Consumption	~ 9267.43 kWh
Comfort	~ 90%

Πίνακας 5.11 Αποτελέσματα αλγορίθμου GreenCapComfort

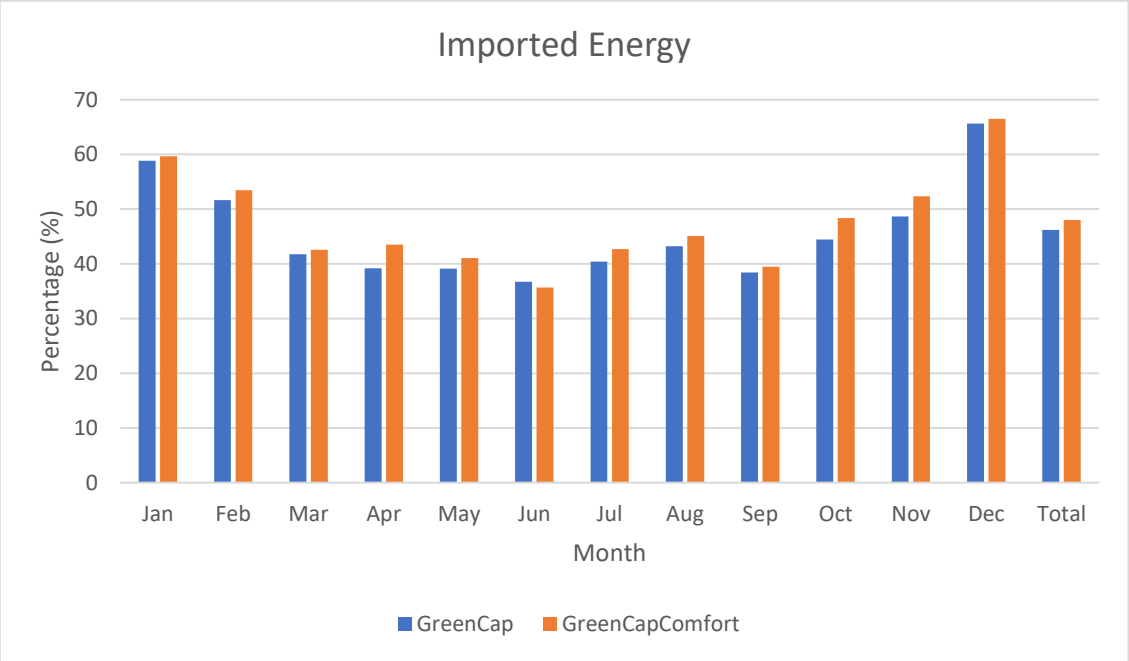
5.3.6 Σύγκριση αλγορίθμου GreenCap με GreenCapComfort

Οι αλγόριθμοι GreenCap και GreenCapComfort ακολουθούν την ίδια λογική του γενετικού αλγόριθμου εκτελώντας ακόμα δύο ευρετικές συναρτήσεις, με την μόνη διαφορά να είναι το fitness function. Η διαφορά στα αποτελέσματα τους είναι ότι στην περίπτωση του GreenCap επιτυγχάνουμε να έχουμε λιγότερη εισαγόμενη ενέργεια από το ενεργειακό δίκτυο, ενώ στον αλγόριθμο GreenCapComfort επιτυγχάνουμε να έχουμε μεγαλύτερη άνεση στον χρήστη, με ένα μικρό κόστος ως προς την εισαγόμενη ενέργεια.

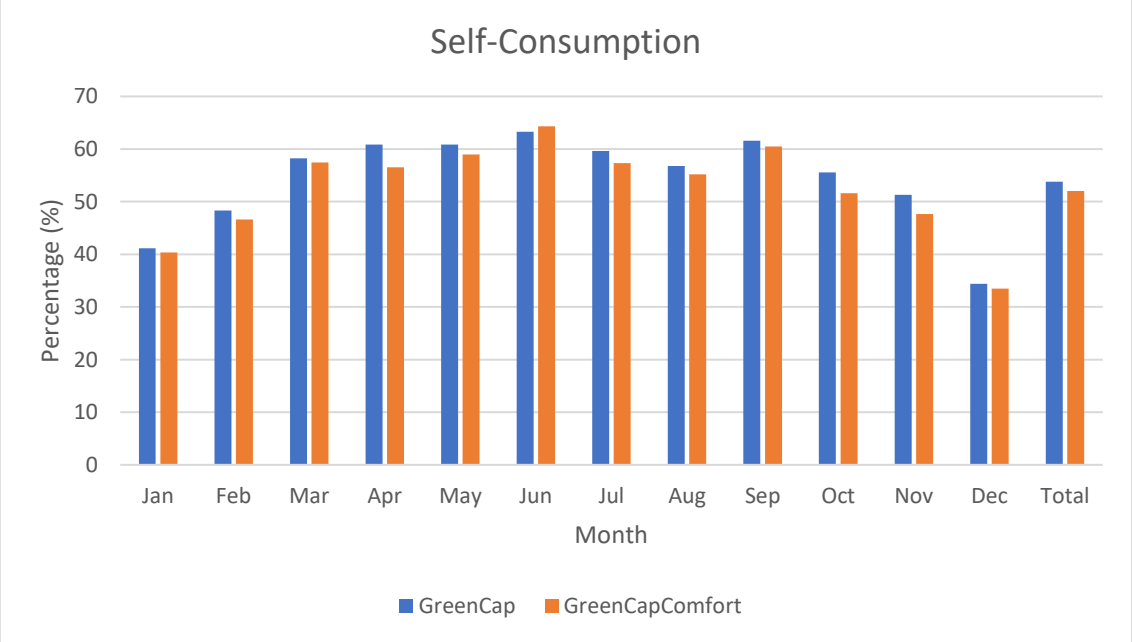
Στο Σχήμα 5.21 παρατηρούμε ότι η εισαγόμενη ενέργεια από το ενεργειακό δίκτυο για κάθε μηνά του χρόνου στον αλγόριθμο GreenCap είναι ελάχιστα μικρότερη από τον αλγόριθμο GreenCapComfort, κάτι που είναι αναμενόμενο. Πιο συγκεκριμένα, η συνολική εισαγόμενη ενέργεια στον αλγόριθμο GreenCap είναι περίπου 46% ενώ στον αλγόριθμο GreenCapComfort είναι περίπου 49%.

Το ίδιο παρατηρούμε και στο Σχήμα 5.22, αφού ο αλγόριθμος GreenCap επιτυγχάνει να καταναλώσει περισσότερη ηλιακή ενέργεια που παράγεται από το φωτοβολταϊκό σύστημα σε σχέση με τον αλγόριθμο GreenCapComfort. Πιο συγκεκριμένα, ο αλγόριθμος GreenCap επιτυγχάνει να ιδιοκαταναλωθεί το 54% της συνολικής

κατανάλωσης ενέργειας, ενώ ο αλγόριθμος GreenCapComfort επιτυγχάνει να ιδιοκαταναλωθεί το 51% της συνολικής κατανάλωσης ενέργειας.



Σχήμα 5.21 Εισαγόμενη ενέργεια αλγορίθμων GreenCap και GreenCapComfort



Σχήμα 5.22 Ιδιοκατανάλωση αλγορίθμων GreenCap και GreenCapComfort

5.3.7 Σύγκριση αποτελεσμάτων

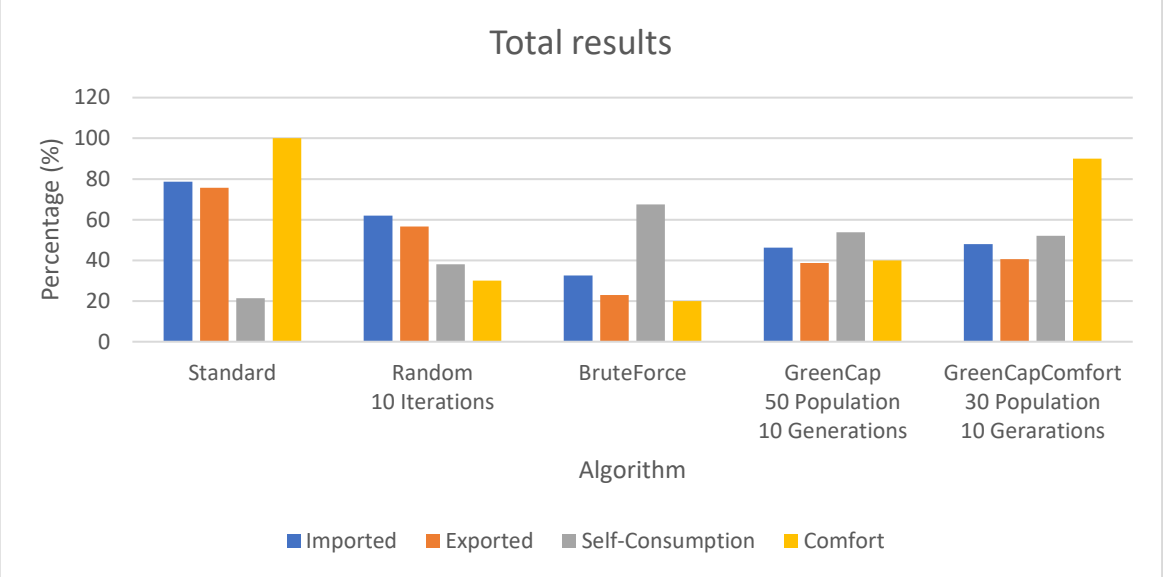
Έχοντας δοκιμάσει διάφορους αλγορίθμους γνωρίζοντας πως ήταν τα αρχικά μας δεδομένα και ποια είναι η βέλτιστη εισαγόμενη ενέργεια που μπορεί να φτάσει κάποιος αλγόριθμος, έχουμε τα ακόλουθα αποτελέσματα όπως φαίνονται στα ακόλουθα σχήματα.

Αρχικά στο Σχήμα 5.23 φαίνεται η ποσοστιαία εισαγόμενη, εξαγόμενη ενέργεια ως προς την συνολική κατανάλωση και την συνολική παραγωγή ενέργειας καθώς και την ιδιοκατανάλωση η οποία γίνεται από τους διάφορους αλγορίθμους. Παρατηρούμε ότι τα καλύτερα αποτελέσματα δίνονται από τον αλγόριθμο Brute Force, κάτι που είναι αναμενόμενο καθώς αλγόριθμος Brute Force δίνει την βέλτιστη χρονοδρομολόγηση των συσκευών. Αναλυτικότερα ο αλγόριθμος Brute Force επιτυγχάνει να μειώσει την εισαγόμενη ενέργεια σε περίπου 32% ως προς την συνολική κατανάλωση, την εξαγόμενη ενέργεια σε περίπου 22% ως προς την συνολική παραγωγή ενέργειας, ενώ το ποσοστό της συνολικής κατανάλωσης ενέργειας το οποίο ιδιοκαταναλώνεται είναι περίπου 68%.

Επίσης παρατηρούμε ότι οι αλγόριθμοι GreenCap και GreenCapComfort δίνουν καλύτερα αποτελέσματα από τον αλγόριθμο Random και από τα κανονικά μας δεδομένα που δίνονται από τον αλγόριθμο Standard κάτι που είναι επιθυμητό. Πιο συγκεκριμένα οι αλγόριθμοι GreenCap και GreenCapComfort επιτυγχάνουν να μειώσουν την εισαγόμενη ενέργεια σε περίπου 46% και 49% αντίστοιχα σε σχέση με 79% που είναι στον αλγόριθμο Standard όπου είναι τα αρχικά μας δεδομένα. Σε σχέση με την βέλτιστη λύση, η εισαγόμενη ενέργεια είναι σε πιο χαμηλά επίπεδα, όμως υπάρχουν κάποια ελαφρυντικά σε σχέση με το Brute Force, όπως για παράδειγμα ο χρόνος εκτέλεσης όπως φαίνεται παρακάτω αλλά και το κόστος της ενέργειας, αφού γίνεται ανακατανομή από τις ώρες αιχμής στους αλγόριθμους GreenCap και GreenCapComfort.

Όσον αφορά το επίπεδο άνεσης του χρήστη και πόσοι από τους κανόνες άνεσης εκτελούνται, παρατηρούμε ξανά στο Σχήμα 5.23 ότι αλγόριθμος GreenCapComfort επιτυγχάνει να έχει την ψηλότερη άνεση σε σχέση με τους υπόλοιπους, εκτός τον Standard κάτι που είναι αναμενόμενο καθώς ο Standard αποτελεί την πραγματική κατανάλωση του χρήστη.

Στο Σχήμα 5.24 φαίνεται οι συνολικές εκπομπές ρύπων διοξειδίου του άνθρακα σε διάφορες χώρες που προκύπτουν χρησιμοποιώντας τον Πίνακα 5.12. Φαίνεται ξεκάθαρα ότι σε χώρες όπως η Σουηδία ή η Λιθουανία δεν υπάρχει ανάγκη να εφαρμοστεί ο αλγόριθμος τον οποίο σχεδιάσαμε. Όμως σε χώρες όπως η Πολωνία ή η Εσθονία, υπάρχει ανάγκη καθώς η συνολική εκπομπή ρύπων CO2 μειώνεται μέχρι και στο 40% με την χρησιμοποίηση των αλγορίθμων GreenCap και GreenCapComfort.

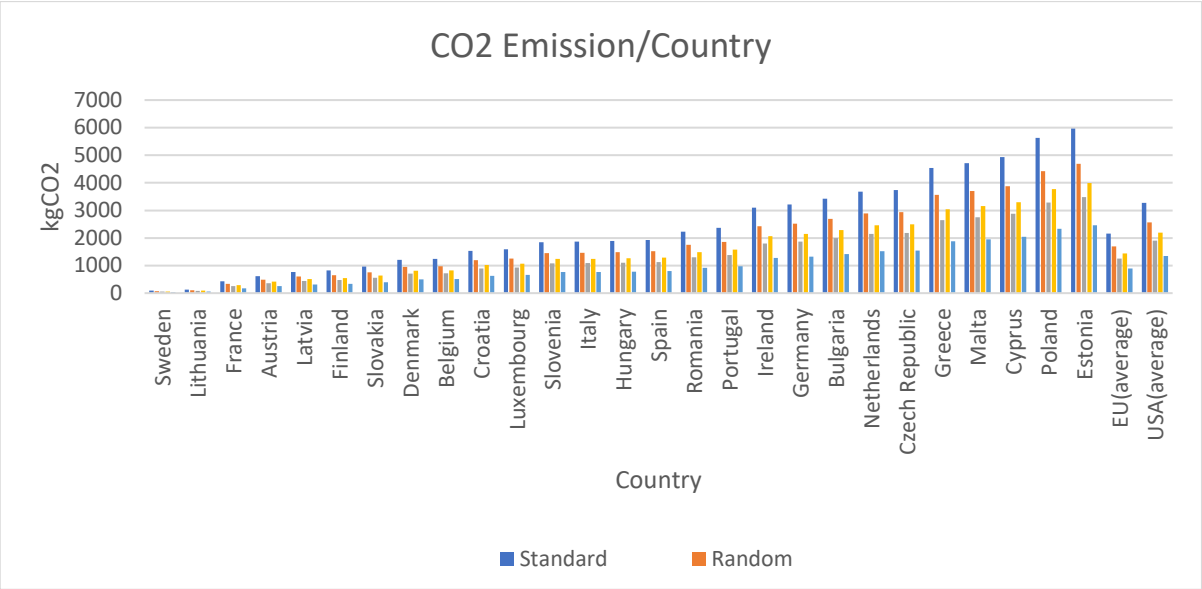


Σχήμα 5.23 Συνολικά αποτελέσματα αλγορίθμων

COUNTRY	kg CO_2 per kWh
Sweden	0.013
Lithuania	0.018
France	0.059
Austria	0.085
Latvia	0.105
Finland	0.113
Slovakia	0.132
Denmark	0.166
Belgium	0.17
Croatia	0.21
Luxembourg	0.219
Slovenia	0.254
Italy	0.256
Hungary	0.26
Spain	0.265

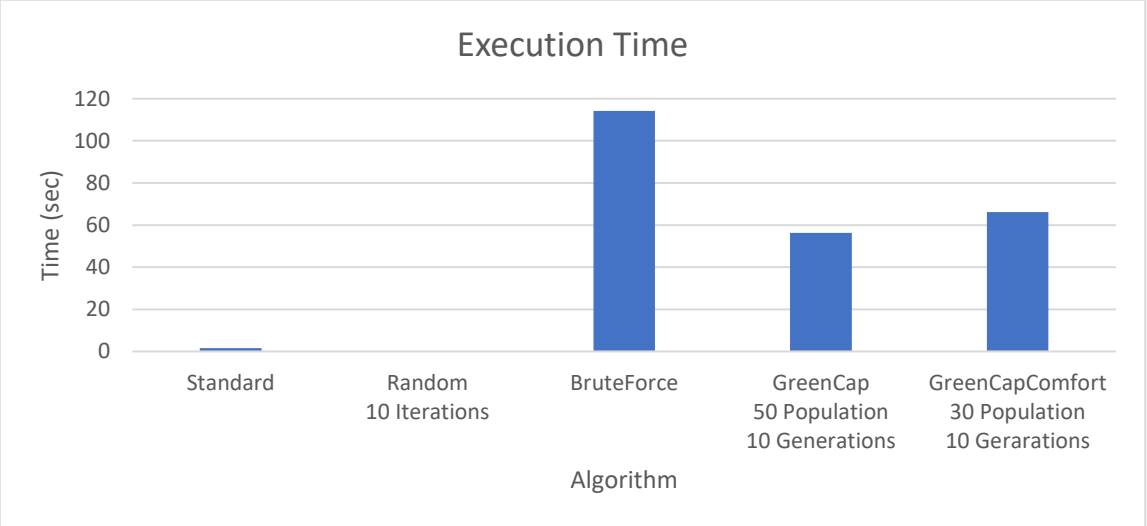
COUNTRY	kg CO_2 per kWh
Romania	0.306
Portugal	0.325
Ireland	0.425
Germany	0.441
Bulgaria	0.47
Netherlands	0.505
Czech Republic	0.513
Greece	0.623
Malta	0.648
Cyprus	0.677
Poland	0.773
Estonia	0.819
COUNTRY	**kg CO_2 per kWh**
EU-27 (average)	0.296
USA (average)	0.449

Πίνακας 5.12 Εκπομπές CO2 ανά kWh σε διάφορες χώρες



Σχήμα 5.24 Εκπομπές ρύπων CO2 ανά αλγόριθμο σε διάφορες χώρες

Στο Σχήμα 5.25 βλέπουμε τον χρόνο εκτέλεσης των αλγορίθμων σε σχέση με τα δεδομένα που δίνονται. Παρατηρούμε ότι ο αλγόριθμος Brute Force χρειάζεται τον περισσότερο χρόνο, κάτι που είναι αναμενόμενο καθώς χρειάζεται περισσότερο χρόνο για να βρεθεί η βέλτιστη λύση. Επίσης ο αλγόριθμος GreenCapComfort χρειάζεται περισσότερο χρόνο από τον αλγόριθμο GreenCap καθώς υπολογίζεται και η άνεση του χρήστη σε αυτήν τον αλγόριθμο. Οι αλγόριθμοι Standard και Random χρειάζονται τον λιγότερο χρόνο εκτέλεσης, όμως τα αποτελέσματα τους δεν είναι καλύτερα από τους υπόλοιπους.



Σχήμα 5.25 Αποτελέσματα αλγορίθμων για τον χρόνο εκτέλεσης

Συνολικά μπορούμε να παρατηρήσουμε ότι οι αλγόριθμοι GreenCap και GreenCapComfort έχουν αρκετά ικανοποιητικά αποτελέσματα σε σύγκριση με τα αρχικά μας δεδομένα από τον αλγόριθμο Standard αλλά και από την βέλτιστη λύση από τον αλγόριθμο Brute Force. Επιτυγχάνουν να υπάρξει μείωση της εισαγόμενης ενέργειας άρα και της εκπομπής ρύπων διοξειδίου του άνθρακα, αλλά και μείωση του κόστους της ενέργειας λόγω της μετατόπισης που γίνεται από τις ώρες αιχμής. Επίσης με τον αλγόριθμο GreenCapComfort επιτυγχάνεται να διατηρηθεί και το επίπεδο άνεσης του χρήστη αρκετά ψηλά, μειώνοντας παράλληλα και την εισαγόμενη ενέργεια.

Κεφάλαιο 6

Συμπεράσματα

Σε αυτό το κεφάλαιο θα αναλυθούν τα συμπεράσματα τα οποία βγήκαν κατά την διάρκεια της Ατομικής Διπλωματικής Εργασίας. Επίσης θα γίνει μία αποτίμηση της δουλειάς της οποίας έγινε τους τελευταίους μήνες και τι έμεινε να γίνει ως μελλοντική δουλειά.

6.1 Συμπεράσματα

Ανακεφαλαιώνοντας, η Ατομική Διπλωματική μου Εργασία είχε ως στόχο να σχεδιαστεί ένας αλγόριθμος ο οποίος θα είχε ως στόχο την χρονοδρομολόγηση των έξυπνων συσκευών σε ένα σπίτι με σκοπό την εκμετάλλευση όσων το δυνατόν περισσότερης ηλιακής ενέργειας η οποία παράγεται από το φωτοβολταϊκό σύστημα το οποίο υπάρχει εγκατεστημένο σε μία οικεία, μειώνοντας παράλληλα την εισαγωγή ενέργειας από το ενεργειακό δίκτυο άρα και τις εκπομπές ρύπων διοξειδίου του άνθρακα (CO2). Επίσης ο αλγόριθμος θα προσπαθεί να εκμεταλλευτεί και το Demand Response έτσι ώστε το κόστος της ενέργειας να είναι όσον το δυνατόν λιγότερο. Επιπλέον αν είναι δυνατόν το επίπεδο άνεση του χρήστη να είναι όσον το δυνατόν μεγαλύτερο. Για την επίλυση του προβλήματος αυτού υλοποιήθηκαν δύο εκδόσεις ενός αλγορίθμου, μία όπου λαμβάνεται υπόψιν το επίπεδο άνεσης του χρήστη και μία όπου δεν λαμβάνεται υπόψιν.

Για την υλοποίηση των δύο εκδόσεων του αλγορίθμου χρησιμοποιήθηκε η λογική του γενετικού αλγορίθμου, και εμπλουτίστηκε με δύο ακόμα ευρετικές συναρτήσεις όπου η κάθε μία έχει ως στόχο να λύσει ένα πρόβλημα όπως αναλύθηκαν προηγουμένως.

Τους πρώτες μήνες της Ατομικής Διπλωματικής μου Εργασίας έγινε αρκετή έρευνα πάνω σε υπάρχοντες έρευνες, δημοσιεύσεις και προϊόντα που υπάρχουν διαθέσιμα. Το πρόβλημα της υπερθέρμανσης του πλανήτη είναι επίκαιρο και ιδιαίτερα σοβαρό, και με την δουλειά η οποία έγινε μπορούμε να βοηθήσουμε και εμείς για την επίλυση του προβλήματος αυτού.

Στην συνέχεια, και μετά την απόφαση ότι θα προχωρήσουμε στην ανάπτυξη γενετικού αλγορίθμου για την επίλυση του προβλήματος το οποίο θέσαμε, ξεκινήσαμε και υλοποιήσαμε τις βασικές διεργασίες ενός γενετικού αλγορίθμου. Μετά την ολοκλήρωση της υλοποίησης των βασικών διεργασιών του γενετικού αλγορίθμου, και μετά από αρκετές δοκιμές, δεν είχαμε τα αναμενόμενα αποτελέσματα.

Ακολούθως υλοποιήσαμε ακόμα μία ευρετική συνάρτηση η οποία τρέχει σε όλες τις γενιές του γενετικού αλγόριθμου, και πιο συγκεκριμένα την συνάρτηση ανακατανομής από τις ώρες αιχμής στις μη ώρες αιχμής. Η συνάρτηση αυτή υλοποιήθηκε για την αξιοποίηση του Demand Response, με στόχο την μείωση του κόστους της ενέργειας σε μία οικία, αλλά και την εύρεση καλύτερης λύσης ανάμεσα στις γενιές του αλγορίθμου. Και πάλι τα αποτελέσματα δεν ήταν τα αναμενόμενα, καθώς παρατηρήσαμε ότι η συνολική κατανάλωση ενέργειας της οικίας μειωνόταν, κάτι που θα οδηγούσε σε μη ικανοποίηση των απαιτήσεων ενέργειας που θα έχει η οικία.

Στην συνέχεια, για την επίλυση του προβλήματος που παρατηρήσαμε σχετικά με την μείωση στην κατανάλωση ενέργειας, υλοποιήσαμε ακόμα μία ευρετική συνάρτηση, όπου η συνάρτηση αυτή επαναφέρει στα αρχικά επίπεδα την κατανάλωση ενέργειας. Η συνάρτηση αυτή επαναφέρει άπληστα την κατανάλωση ενέργειας στα αρχικά επίπεδα, δηλαδή αν χρειάζεται να αυξήσει την ενέργεια, την αυξάνει στις ώρες που υπάρχει περισσότερη παραγωγή ενέργειας, και αν χρειάζεται να μειώσει ενέργεια, μειώνει στις ώρες που δεν υπάρχει παραγωγή.

Ακολούθως και με την δοκιμή του αλγορίθμου με διάφορες παραμέτρους, και την εύρεση των βέλτιστων παραμέτρων ως προς το καλύτερο αποτέλεσμα, δηλαδή την ελαχιστοποίηση της εισαγόμενης ενέργειας άρα και την εκπομπή ρύπων διοξειδίου του

άνθρακα (CO2), αλλά και ως προς τον χρόνο εκτέλεσης του αλγορίθμου προσθέσαμε και το επίπεδο άνεσης του χρήστη στον αλγόριθμο.

Για να μετρήσουμε το επίπεδο άνεσης του χρήστη, χρησιμοποιήσαμε κάποιους κανόνες άνεσης, όπου αυτοί οι κανόνες άνεσης θα δίνονται από τον χρήστη. Παρατηρήσαμε ότι τα αποτελέσματα του αλγορίθμου που σχεδιάσαμε ως προς το επίπεδο άνεσης του χρήστη δεν ήταν αρκετά καλά (περίπου 30-40%). Για την επίλυση του προβλήματος αυτού, υλοποιήσαμε και μία δεύτερη έκδοση του αλγορίθμου όπου την ονομάσαμε GreenCapComfort, όπου λαμβάνεται υπόψιν και το επίπεδο άνεσης του χρήστη. Σε αυτήν την έκδοση, πέτυχαμε την αύξηση του επιπέδου άνεσης του χρήστη (περίπου 90%) όμως με μία μικρή θυσία στην εισαγόμενη ενέργεια (περίπου 5% αύξηση σε σχέση με το GreenCap).

Στο τέλος των πειραμάτων τα οποία κάναμε συγκρίναμε τις δύο εκδόσεις του αλγορίθμου με τα αρχικά μας δεδομένα (Standard algorithm), την τυχαία λύση (Random algorithm) και την βέλτιστη λύση (Brute Force algorithm). Παρατηρήσαμε ότι πετύχαμε να ανεβάσουμε την ιδιοκατανάλωση η οποία γίνετε στην οικία από 21% στα αρχικά μας δεδομένα σε περίπου 50 – 54% στις δύο εκδόσεις του αλγορίθμου, και σε σχέση με την βέλτιστη λύση όπου είναι στο 68% είναι αρκετά ικανοποιητικό. Επίσης καταφέραμε να μειώσουμε την εισαγόμενη ενέργεια από περίπου 78% που χρειάζεται σε σχέση με την συνολική κατανάλωση της οικίας, σε περίπου 46-49%. Επιπλέον με την χρήση της έκδοσης του GreenCapComfort πετύχαμε να έχουμε την μείωση στην εισαγόμενη ενέργεια χωρίς να επηρεαστεί ιδιαίτερα το επίπεδο άνεσης του χρήστη καθώς πετύχαμε να το διατηρήσουμε κοντά στο 90%.

6.2 Μελλοντική Δουλειά

Ως μελλοντική δουλειά έμεινε η πλήρης ενσωμάτωση των δύο αλγορίθμων σε ένα controller έτσι ώστε να δοκιμαστεί σε πραγματικό περιβάλλον ο αλγόριθμος, και η θεωρητική επιτυχία του αλγορίθμου στην μείωση της εισαγόμενης ενέργειας από το ενεργειακό δίκτυο να γίνει και πρακτική. Για την ενσωμάτωση του αλγορίθμου μπορεί να χρησιμοποιήσει είτε το μοντέλο του OpenHAB είτε το μοντέλο του Domoticz. Επίσης με την ενσωμάτωση του αλγόριθμου με ένα από αυτά τα συστήματα θα μας δίνει την

δυνατότητα να δοκιμαστεί ο αλγόριθμος σε διαφορετικά σε μέγεθος κτήρια, όπως σε ένα διαμέρισμα, σε μία οικία ή ακόμα και σε ένα μπλοκ διαμερισμάτων.

Επίσης στον αλγόριθμο μπορούν να γίνουν και κάποιες επεκτάσεις που θα έχουν ως στόχο ο αλγόριθμος να είναι πιο αποδοτικός και πιο αποτελεσματικός. Για παράδειγμα θα μπορούσαν να γίνουν κάποιες αλλαγές στις διαδικασίες του γενετικού αλγορίθμου, όπως την χρησιμοποίηση δύο crossover σημείων στην διαδικασία της διασταύρωσης των δύο λύσεων. Ακόμη στην διαδικασία της μετάλλαξης θα μπορούσαν να γίνουν κάποιες αλλαγές, όπως για παράδειγμα να μην σβήνουν μόνο η συσκευές στο σημείο μετάλλαξης, αλλά και η κατανάλωση της ενέργειας στο σημείο αυτό να είναι ίση με την μέση ή την μέγιστη για την κάθε συσκευή. Επιπλέον θα μπορούσε να υλοποιηθεί ένα μοντέλο μηχανικής μάθησης όπου θα εκπαιδευόταν με τα δεδομένα που χρησιμοποιήσαμε, με στόχο την χρησιμοποίηση του μοντέλου αυτού για την εξαγωγή κανόνων άνεσης χωρίς να τις δίνει ο χρήστης, έτσι ώστε να γίνει πιο εύκολη η χρησιμοποίηση του συστήματος το οποίο προτείνουμε.

Αναφορές

[1] “Causes of climate change,” Climate Action - European Commission, 28-Jun-2017. [Online]. Available: https://ec.europa.eu/clima/change/causes_en. [Accessed: 05-Oct-2021].

[2] R. Samuel Selvaraj and V. Sivamadhavi, "Magnitude of Green House Effect and the contribution of Carbon di oxide," Recent Advances in Space Technology Services and Climate Change 2010 (RSTS & CC-2010), 2010, pp. 41-44, doi: 10.1109/RSTSCC.2010.5712795.

[3] Grzegorz Bazydlo and Szymon Werminski, “Demand side management through home area network systems,” International Journal of Electrical Power and Energy Systems, 2018, pp. 174-185

[4] “Greenhouse Gas Emission Intensity of Electricity Generation in Europe.” Www.eea.europa.eu, www.eea.europa.eu/ims/greenhouse-gas-emission-intensity-of-1.

[5] “Climate Strategies & Targets.” Ec.europa.eu, ec.europa.eu/clima/eu-action/climate-strategies-targets_en.

[6] “Peak Demand.” Wikipedia, 12 June 2021, en.wikipedia.org/wiki/Peak_demand.

[7] “Global IoT Market Size Grew 22% in 2021 — These 16 Factors Affect the Growth Trajectory to 2027.” IoT Analytics, 30 Mar. 2022, iot-analytics.com/iot-market-size/.

[8] “IoT and the Environment | IEEE Communications Society.” Www.comsoc.org, www.comsoc.org/publications/magazines/ieee-internet-things-magazine/cfp/iot-and-environment.

[9] “What We Can Do.” Center for Climate and Energy Solutions, 5 Oct. 2017, www.c2es.org/content/what-we-can-do/.

[10] AG, SMA Solar Technology. “Systematic and Intelligent Energy Management» Home | SMA Solar.” Www.sma.de, www.sma.de/en/home/systematic-intelligent-energy-management.html. Accessed 17 May 2022.

[11] “Energy Manager.” Bosch Global, www.bosch.com/stories/smart-home-energy-management-system/.

[12] “Nest Learning Thermostat - Programs Itself Then Pays for Itself - Google Store.” Google.com, 2009, store.google.com/us/product/nest_learning_thermostat_3rd_gen?hl=en-US.
[13] “Wi-Fi Smart Thermostat | Ecobee.” Www.ecobee.com, www.ecobee.com/en-ca/smart-thermostats/smart-wifi-thermostat/.
[14] “What Is a Smart Appliance - and How Do You Make One?” Www.aylanetworks.com, www.aylanetworks.com/blog/what-is-a-smart-appliance-and-how-do-you-make-one.
[15] “Wemo WiFi Smart Plug for Alexa, Google, HomeKit.” Belkin, www.belkin.com/us/smart-home/wemo/wemo-wifi-smart-plug/p/p-wsp080/. Accessed 17 May 2022.
[16] “OpenHAB.” Www.openhab.org, www.openhab.org/.
[17] “Domoticz.” Domoticz.com, 2012, domoticz.com/. Accessed 17 May 2022.
[18] “IOS App.” Www.openhab.org, www.openhab.org/docs/apps/ios.html. Accessed 17 May 2022.
[19] “Domoticz - Home Automation.” App Store, apps.apple.com/us/app/domoticz-home-automation/id1251318855?platform=iphone. Accessed 17 May 2022.
[20] Wikipedia Contributors. “Energy Demand Management.” Wikipedia, Wikimedia Foundation, 26 May 2019, en.wikipedia.org/wiki/Energy_demand_management.
[21] “Demand Response.” Energy.gov, www.energy.gov/oe/activities/technology-development/grid-modernization-and-smart-grid/demand-response.
[22] “The Load Shifting Low-Down: Your Quick Guide.” GridBeyond, 19 Oct. 2018, gridbeyond.com/load-shifting-low-down-guide. Accessed 17 May 2022.
[23] “Figure 2.2: Concepts of Load Curtailment and Load Shifting.” ResearchGate, www.researchgate.net/figure/Concepts-of-load-curtailment-and-load-shifting_fig1_337023176. Accessed 17 May 2022.
[24] “Linear Programming | Brilliant Math & Science Wiki.” Brilliant.org, 2019, brilliant.org/wiki/linear-programming/.
[25] Wikipedia Contributors. “Linear Programming.” Wikipedia, Wikimedia Foundation, 29 Sept. 2019, en.wikipedia.org/wiki/Linear_programming.
[26] E. Bejoy, S. N. Islam and A. M. T. Oo, "Optimal scheduling of appliances through residential energy management," 2017 Australasian Universities Power Engineering Conference (AUPEC), 2017, pp. 1-6, doi: 10.1109/AUPEC.2017.8282505.

[27] Wikipedia Contributors. “Machine Learning.” Wikipedia, Wikimedia Foundation, 29 Apr. 2019, en.wikipedia.org/wiki/Machine_learning.
[28] Neutelings, Izaak. Neural Networks. tikz.net/neural_networks/.
[29] M. S. Ahmed, A. Mohamed, H. Shareef, R. Z. Homod and J. A. Ali, "Artificial neural network based controller for home energy management considering demand response events," 2016 International Conference on Advances in Electrical, Electronic and Systems Engineering (ICAEES), 2016, pp. 506-509, doi: 10.1109/ICAEES.2016.7888097.
[30] D. Zhang, S. Li, M. Sun and Z. O’Neill, "An Optimal and Learning-Based Demand Response and Home Energy Management System," in IEEE Transactions on Smart Grid, vol. 7, no. 4, pp. 1790-1801, July 2016, doi: 10.1109/TSG.2016.2552169.
[31] “Integer Linear Programming | NEOS.” Neos-Guide.org, neos-guide.org/content/integer-linear-programming.
[32] O. Erdinc, N. G. Paterakis, T. D. P. Mendes, A. G. Bakirtzis and J. P. S. Catalão, "Smart Household Operation Considering Bi-Directional EV and ESS Utilization by Real-Time Pricing-Based DR," in IEEE Transactions on Smart Grid, vol. 6, no. 3, pp. 1281-1291, May 2015, doi: 10.1109/TSG.2014.2352650.
[33] I. -Y. Joo and D. -H. Choi, "Distributed Optimization Framework for Energy Management of Multiple Smart Homes With Distributed Energy Resources," in IEEE Access, vol. 5, pp. 15551-15560, 2017, doi: 10.1109/ACCESS.2017.2734911.
[34] “Mixed Integer Nonlinear Programming | NEOS.” Neos-Guide.org, neos-guide.org/content/mixed-integer-nonlinear-programming.
[35] S. Althaher, P. Mancarella and J. Mutale, "Automated Demand Response From Home Energy Management System Under Dynamic Pricing and Power and Comfort Constraints," in IEEE Transactions on Smart Grid, vol. 6, no. 4, pp. 1874-1883, July 2015, doi: 10.1109/TSG.2014.2388357.
[36] “Knapsack Problem.” Wikipedia, 5 Sept. 2020, en.wikipedia.org/wiki/Knapsack_problem.
[37] M. Anzar, R. Iqra, A. Kousar, S. Ejaz, M. S. Alvarez-Alvarado and A. K. Zafar, "Optimization of Home Energy Management System in Smart Grid for Effective Demand Side Management," 2017 International Renewable and Sustainable Energy Conference (IRSEC), 2017, pp. 1-6, doi: 10.1109/IRSEC.2017.8477255.

[38] S. Rahim, S. A. Khan, N. Javaid, N. Shaheen, Z. Iqbal and G. Rehman, "Towards Multiple Knapsack Problem Approach for Home Energy Management in Smart Grid," 2015 18th International Conference on Network-Based Information Systems, 2015, pp. 48-52, doi: 10.1109/NBiS.2015.11.
[39] Vijini Mallawaarachchi. “Introduction to Genetic Algorithms — Including Example Code.” Towards Data Science, Towards Data Science, 8 July 2017, towardsdatascience.com/introduction-to-genetic-algorithms-including-example-code-e396e98d8bf3.
[40] “Figure 1. Flowchart of the Standard Genetic Algorithm (GA) [33].” ResearchGate, www.researchgate.net/figure/Flowchart-of-the-standard-genetic-algorithm-GA-33_fig1_344832850. Accessed 17 May 2022.
[41] M. Awais, M. B. Rasheed, M. Z. Iqbal and M. Usman, "Multiple user based residential energy management scheme for smart homes," 2018 1st International Conference on Power, Energy and Smart Grid (ICPESG), 2018, pp. 1-6, doi: 10.1109/ICPESG.2018.8384492.
[42] A. Imran et al., "An Optimal Energy Management Strategy Under Hybrid Generation and Price-based Demand Response Program in Smart Grid," 2020 2nd International Conference on Smart Power & Internet Energy Systems (SPIES), 2020, pp. 548-553, doi: 10.1109/SPIES48661.2020.9243123.
[43] “What Is Dynamic Programming? - Grokking Dynamic Programming Patterns for Coding Interviews.” Educative: Interactive Courses for Software Developers, www.educative.io/courses/grokking-dynamic-programming-patterns-for-coding-interviews/m2G1pAq0OO0.
[44] C. Xia, W. Li, X. Chang, F. C. Delicato, T. Yang and A. Y. Zomaya, "Edge-based Energy Management for Smart Homes," 2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/DataCom/CyberSciTech), 2018, pp. 849-856, doi: 10.1109/DASC/PiCom/DataCom/CyberSciTec.2018.00-19.
[45] S. Constantinou, A. Konstantinidis, D. Zeinalipour-Yazti, and P. K. Chrysanthis, “The iot meta-control firewall,” in 37th IEEE International Conference on Data Engineering, ser. ICDE‘21. April 19 - April 22, 2021, Chania, Crete: IEEE Computer Society, 2021.

[46] S. Constantinou, A. Konstantinidis and D. Zeinalipour-Yazti, "Green Planning Systems for Self-Consumption of Renewable Energy," in IEEE Internet Computing, doi: 10.1109/MIC.2022.3164581.

[47] Υπό δημοσιέυση: Soteris Constantinou, Andreas Konstantinidis, Panos K. Chrysanthis, and Demetrios Zeinalipour-Yazti. 2022. Green Planning of IoT Home Automation Workflows in Smart Buildings. ACM Trans. Internet Things 1, 1 (April 2022)

[48] “Genetic Algorithms.” GeeksforGeeks, 29 June 2017, www.geeksforgeeks.org/genetic-algorithms/.

[49] “Crossover in Genetic Algorithm.” GeeksforGeeks, 21 June 2019, www.geeksforgeeks.org/crossover-in-genetic-algorithm/.

[50] “Smart - UMass Trace Repository.” Traces.cs.umass.edu, traces.cs.umass.edu/index.php/smart/smart. Accessed 17 May 2022.

[51] “Real-Time Operating Grid - U.S. Energy Information Administration (EIA).” Www.eia.gov, www.eia.gov/electricity/gridmonitor/dashboard/electric_overview/US48/US48.

[52] “Dynamic Programming for Fun — Nimbus: Maximize Your Programmatic Ad Revenue.” Nimbus: Maximize Your Programmatic Ad Revenue, www.adsbynimbus.com/tech-blog/dynamic-programming. Accessed 17 May 2022.

Παράρτημα Α

```java
package cs.ucy.ac.cy;

import java.io.IOException;

public class Main {

    public static void main(String[] args) throws IOException, InterruptedException {

        // set the input variable as dev for development environment or as prod for production environment
        // and create data object
        ImportedData data = ImportedData.importData("win");

        // if 1 it will get rules from API else it will get rules from file
        int getFromAPI = 0;
        data.retrieveRules(getFromAPI);

        // if method = 1 the functions will run with real consumption
        // else the functions will run with binary (on/off) data
        int method = 1;

        // start the program
        App.start(data, method);

    }
}
```

```java
package cs.ucy.ac.cy;

import com.opencsv.CSVReader;
import com.opencsv.CSVReaderBuilder;

import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

// object to save the data
public class ImportedData {

    // object variables
    private final List<String[]> peakHours;
    private List<String[]> energyConsumption;
    private final List<String[]> solarGen;
    private List<String[]> hourlyEnergyConsumption;
    private List<String[]> binaryEnergyConsumption;
    private List<String[]> rules;

    // object constructor
    public ImportedData() {
        this.peakHours = null;
        this.energyConsumption = null;
        this.solarGen = null;
        this.hourlyEnergyConsumption = null;
        this.binaryEnergyConsumption = null;
        this.rules = null;
    }

    // object constructor
    public ImportedData(List<String[]> peakHours, List<String[]> energyConsumption, List<String[]> solarGen) {
        this.peakHours = peakHours;
        this.energyConsumption = energyConsumption;
        this.solarGen = solarGen;
        this.hourlyEnergyConsumption = null;
        this.binaryEnergyConsumption = null;
```

```
        this.rules = null;
    }

    // get peak hours list
    public List<String[]> getPeakHours() { return peakHours; }

    // get energy consumption list
    public List<String[]> getEnergyConsumption() { return energyConsumption; }

    // get solar generation list
    public List<String[]> getSolarGen() { return solarGen; }

    // get appliance energy consumption per hour
    public List<String[]> getHourlyEnergyConsumption() { return hourlyEnergyConsumption; }

    // set appliance energy consumption per hour
    public void setHourlyEnergyConsumption(List<String[]> hourlyEnergyConsumption) {
        this.hourlyEnergyConsumption = hourlyEnergyConsumption;
    }

    // get appliance energy consumption per hour (on/off)
    public List<String[]> getBinaryEnergyConsumption() { return binaryEnergyConsumption; }

    // set appliance binary energy consumption per hour (on/off)
    public void setBinaryEnergyConsumption(List<String[]> binaryEnergyConsumption) {
        this.binaryEnergyConsumption = binaryEnergyConsumption;
    }

    public List<String[]> getRules() { return rules; }

    public static ImportedData importData(String env) {

        // paths for the data according to the environment
        String peakHoursPath = env.equals("dev")
                ? "/Users/nicolaspolycarpou/Documents/GreenCap/Dataset/US_Peak_Hours/EIA930_BALANCE_2016_NYIS.csv"
                : env.equals("prod")
                ? "./Temp/Space"
                : env.equals("win")
                ? "C:\\Users\\CPS\\Desktop\\GreenCap\\Data\\EIA930_BALANCE_2016_NYIS.csv"
                : ".";
        String energyConsumptionPath = env.equals("dev")
                ? "/Users/nicolaspolycarpou/Documents/GreenCap/Dataset/Home_Data_fixed/HomeEnergyConsumption2016.csv"
                : env.equals("prod")
                ? "./Temp/Space"
                : env.equals("win")
                ? "C:\\Users\\CPS\\Desktop\\GreenCap\\Data\\HomeEnergyConsumption2016.csv"
                : ".";
        String solarGenPath = env.equals("dev")
                ? "/Users/nicolaspolycarpou/Documents/GreenCap/Dataset/Home_Solar_Gen/Solar_Data.csv"
                : env.equals("prod")
                ? "./Temp/Space"
                : env.equals("win")
                ? "C:\\Users\\CPS\\Desktop\\GreenCap\\Data\\Solar_Data.csv"
                : ".";

        ImportedData data = new ImportedData();

        try {

            // filereader to read the data
            FileReader peakHoursFilereader = new FileReader(peakHoursPath);
            FileReader energyConsumptionFilereader = new FileReader(energyConsumptionPath);
            FileReader solarGenFilereader = new FileReader(solarGenPath);

            // use csvreader to read the csv files
            CSVReader peakHoursCsvReader = new CSVReaderBuilder(peakHoursFilereader)
                    .withSkipLines(1)
                    .build();
            CSVReader energyConsumptionCsvReader = new CSVReaderBuilder(energyConsumptionFilereader)
                    .withSkipLines(1)
                    .build();
            CSVReader solarGenCsvReader = new CSVReaderBuilder(solarGenFilereader)
                    .withSkipLines(1)
                    .build();

            // create the data object
            data = new ImportedData(peakHoursCsvReader.readAll(),
                    energyConsumptionCsvReader.readAll(), solarGenCsvReader.readAll());
```

```
        } catch (Exception e) {
            System.out.println("File not found!!!");
        }

        return data;
    }

    public void reallocationToNonProduction() throws IOException {

        // list to save output
        List<String[]> output = new ArrayList<>();

        // iterate day by day
        for (int i = 0; i < this.energyConsumption.size(); i+=(24*60)){

            String[][] day = new String[24*60][20];
            for (int j = i; j < i + (10*60); j++){
                System.arraycopy(this.energyConsumption.get(j),
                        srcPos: 0, day[j % (24 * 60)],  destPos: 0, day[j % (24 * 60)].length);
            }
            for (int j = i + (10*60); j < (i + 15*60); j++){
                day[j % (24*60)][0] = this.energyConsumption.get(j)[0];
                for (int k = 1; k < day[j % (24*60)].length; k++){
                    if (Float.parseFloat(this.energyConsumption.get(j)[k])
                            < Float.parseFloat(this.energyConsumption.get(j + (7*60))[k]))
                        day[j % (24*60)][k] = this.energyConsumption.get(j)[k];
                    else
                        day[j % (24*60)][k] = this.energyConsumption.get(j + (7*60))[k];
                }
            }
            for (int j = i + (15*60); j < (i + 17*60); j++){
                System.arraycopy(this.energyConsumption.get(j),
                        srcPos: 0, day[j % (24 * 60)],  destPos: 0, day[j % (24 * 60)].length);
            }
            for (int j = i + (17*60); j < (i + 22*60); j++){
                day[j % (24*60)][0] = this.energyConsumption.get(j)[0];
                for (int k = 1; k < day[j % (24*60)].length; k++){
                    if (Float.parseFloat(this.energyConsumption.get(j)[k])
                            > Float.parseFloat(this.energyConsumption.get(j - (7*60))[k]))
                        day[j % (24*60)][k] = this.energyConsumption.get(j)[k];
                    else
                        day[j % (24*60)][k] = this.energyConsumption.get(j - (7*60))[k];
                }
            }
            for (int j = i + (22*60); j < (i + 24*60); j++){
                System.arraycopy(this.energyConsumption.get(j),
                        srcPos: 0, day[j % (24 * 60)],  destPos: 0, day[j % (24 * 60)].length);
            }

            Collections.addAll(output, day);
        }

        // write results in csv file
        File applianceFile = new File( pathname: "HomeEnergyConsumption2016.csv");
        //applianceFile.createNewFile();
        FileWriter applianceWriter = new FileWriter(applianceFile);
        for (String[] strings : output) {
            for (int j = 0; j < strings.length - 1; j++) {
                applianceWriter.write( str: strings[j] + ",");
            }
            applianceWriter.write( str: strings[strings.length - 1] + "\n");
        }
        applianceWriter.close();

        this.energyConsumption = output;
    }

    public void retrieveRules(int getFromAPI) throws IOException, InterruptedException {

        if (getFromAPI == 1){

            // get rules from the api
            this.rules = ApiCalls.getRulesFromAPI();

            // write rules to csv file
            File rulesFile = new File( pathname: "rules.csv");
            FileWriter rulesWriter = new FileWriter(rulesFile);
            rulesWriter.write( str: "Date & Time,Appliance\n");
            for (String[] rule : rules) {
                rulesWriter.write( str: rule[0] +  "," + rule[1] + "\n");
```

```
        }
        applianceWriter.close();

        this.energyConsumption = output;
    }

    public void retrieveRules(int getFromAPI) throws IOException, InterruptedException {

        if (getFromAPI == 1){

            // get rules from the api
            this.rules = ApiCalls.getRulesFromAPI();

            // write rules to csv file
            File rulesFile = new File( pathname: "rules.csv");
            FileWriter rulesWriter = new FileWriter(rulesFile);
            rulesWriter.write( str: "Date & Time,Appliance\n");
            for (String[] rule : rules) {
                rulesWriter.write( str: rule[0] +  "," + rule[1] + "\n");
            }
            rulesWriter.close();
        }

        // read rules from file
        else {
            String rulesPath = "C:\\Users\\CPS\\Desktop\\GreenCap\\Data\\rules.txt";
            FileReader rulesFilereader = new FileReader(rulesPath);
            CSVReader rulesCsvReader = new CSVReaderBuilder(rulesFilereader)
                .withSkipLines(1)
                .build();
            this.rules = rulesCsvReader.readAll();
        }
    }
}
```

```
package cs.ucy.ac.cy;

import java.io.IOException;
import java.util.Arrays;
import java.util.List;

public class App {

    public static void start(ImportedData data, int method) throws IOException, InterruptedException {

        // make minor changes to dataset from peak to non-peak hour
        //data.reallocationToNonProduction();

        long startTime = System.nanoTime();
        // get total/appliance consumption per hour
        System.out.println("Hourly total/per appliance energy consumption are written in files");
        Statistics.ConsumptionPerHour(data);

        // calculate average consumption for each appliance per hour
        float[] averageConsumptions = Statistics.applianceAverageConsumptionPerHour(data.getHourlyEnergyConsumption());

        System.out.println();
        System.out.println("Average consumption per appliance:");
        System.out.println(Arrays.toString(averageConsumptions));

        // find the max consumption a device can consume in an hour
        float[] maxConsumptions = Statistics.applianceMaxConsumption(data.getHourlyEnergyConsumption());

        System.out.println();
        System.out.println("Max consumption per appliance:");
        System.out.println(Arrays.toString(maxConsumptions));

        if (method == 1) {

            System.out.println("\n*************************************************************************************");

```

```
        System.out.println();
        System.out.println("Statistics:");
        Statistics.redGreenEnergyPerDay(data.getHourlyEnergyConsumption(), averageConsumptions,
                check: 1, data.getRules(), startTime, function: "Standard");

        startTime = System.nanoTime();
        int iterations = 10;
        List<String[]> randomData = Algorithms.random(iterations, data.getHourlyEnergyConsumption(),
                maxConsumptions, averageConsumptions);

        // get red/green energy per hour after random
        System.out.println();
        System.out.println("Statistics after random allocation(Iterations = " + iterations + "):");
        Statistics.redGreenEnergyPerDay(randomData, averageConsumptions, check: 1,
                data.getRules(), startTime, function: "Random");

        // call brute force allocation
        startTime = System.nanoTime();
        List<String[]> bruteForceData = Algorithms.bruteForceSolution(data.getHourlyEnergyConsumption(),
                maxConsumptions);

        // get red/green energy per hour after random
        System.out.println();
        System.out.println("Statistics after brute force allocation(optimal schedule with stacking consumptions):");
        Statistics.redGreenEnergyPerDay(bruteForceData, averageConsumptions, check: 1,
                data.getRules(), startTime, function: "BruteForce");

        // call greencap
        startTime = System.nanoTime();
        List<String[]> genAlgData = GreenCap.GeneticAlgorithm(data.getHourlyEnergyConsumption(),
                data.getPeakHours(), maxConsumptions, averageConsumptions);


        System.out.println();
        System.out.println("Statistics after GreenCap Algorithm(Genetic Algorithm + " +
                "FixConsumption Heuristic + Peak Hours Reallocation Heuristic):");
        Statistics.redGreenEnergyPerDay(genAlgData, averageConsumptions, check: 1,
                data.getRules(), startTime, function: "GreenCap");

        // call greencap comfort
        startTime = System.nanoTime();
        List<String[]> genAlgComfortData =
                GreenCapComfort.GeneticAlgorithmComfort(data.getHourlyEnergyConsumption(),
                        data.getPeakHours(), maxConsumptions, averageConsumptions, data.getRules());

        System.out.println();
        System.out.println("Statistics after GreenCap Algorithm Comfort(Genetic Algorithm + " +
                "FixConsumption Heuristic + Peak Hours Reallocation Heuristic + " +
                "Considering Comfort in Fitness Function):");
        Statistics.redGreenEnergyPerDay(genAlgComfortData, averageConsumptions, check: 1,
                data.getRules(), startTime, function: "GreenCapComfort");
    }
```

```java
package cs.ucy.ac.cy;

import java.io.*;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.*;

public class Statistics {

    // function to create csv file appliance and total energy consumption per hour
    public static void ConsumptionPerHour(ImportedData data) throws IOException {

        List<String[]> energyConsumptionData = data.getEnergyConsumption();
        List<String[]> solarGenData = data.getSolarGen();

        // date formatter to read from data
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");

        // lists to save the data needed
        List<LocalDateTime> hours = new ArrayList<>();
        List<float[]> energyConsumptions = new ArrayList<>();

        // variables to save appliance consumption and current hour
        int currentHour = 0;
        float[] consumptionPerHour = new float[19];

        // for each sample of data
        for (int i = 0; i < energyConsumptionData.size(); i++) {

            // get sample and read hour
            String[] metric = energyConsumptionData.get(i);
            LocalDateTime date = LocalDateTime.parse(metric[0], formatter);
            int readHour = date.getHour();

            // if it is different from previous read save the total consumption and the hour
            if (readHour != currentHour) {

                // save consumption for previous hour
                LocalDateTime oneHourBack = date.minusHours(1);
                hours.add(oneHourBack);
                energyConsumptions.add(consumptionPerHour);

                // set consumption to 0
                consumptionPerHour = new float[19];
                Arrays.fill(consumptionPerHour,  val: 0.0F);

                // set current hour to the hour that is read
                currentHour = readHour;
            }

            // calculate the appliance consumption for this sample
            for (int j = 1; j < metric.length; j++) {
                consumptionPerHour[j - 1] = consumptionPerHour[j - 1] + Float.parseFloat(metric[j]);
            }

            // if is the last sample save the total consumption
            if (i == (energyConsumptionData.size() - 1)) {
                LocalDateTime oneHourBack = date.minusMinutes(59);
                hours.add(oneHourBack);
                energyConsumptions.add(consumptionPerHour);
            }
        }

        // get results in list of string[]
        List<String[]> applianceEnergyConsumptionsPerHour = new ArrayList<>();
        for (int i = 0; i < hours.size(); i++) {
            String[] array = new String[21];
            array[0] = hours.get(i).format(formatter);
            for (int j = 0; j < energyConsumptions.get(i).length; j++) {
                array[j + 1] = String.valueOf(energyConsumptions.get(i)[j]);
            }
            array[20] = solarGenData.get(i)[1];
            applianceEnergyConsumptionsPerHour.add(array);
        }

```

```
        // write result in data object
        data.setHourlyEnergyConsumption(applianceEnergyConsumptionsPerHour);

        // write results in txt file
        File applianceFile = new File( pathname: "ApplianceConsumptionPerHour.csv");
        //applianceFile.createNewFile();
        FileWriter applianceWriter = new FileWriter(applianceFile);
        for (int i = 0; i < hours.size(); i++) {
            applianceWriter.write( str: hours.get(i).format(formatter) + ",");
            float[] row = energyConsumptions.get(i);
            for (float v : row) {
                applianceWriter.write( str: v + ",");
            }
            applianceWriter.write( str: solarGenData.get(i)[1] + "\n");
        }
        applianceWriter.close();

        // write results in csv file
        File totalFile = new File( pathname: "TotalConsumptionPerHour.csv");
        //totalFile.createNewFile();
        FileWriter totalWriter = new FileWriter(totalFile);
        for (int i = 0; i < energyConsumptions.size(); i++) {
            float[] row = energyConsumptions.get(i);
            float total = 0.0F;
            for (float v : row) {
                total = total + v;
            }
            totalWriter.write( str: hours.get(i).format(formatter) + "," + total + "," + solarGenData.get(i)[1] + "\n");
        }
        totalWriter.close();
    }

    // function to calculate red/green energy per hour
    public static void redGreenEnergyPerDay(List<String[]> totalConsumption,
                                            float[] averageConsumption, int check, List<String[]> rules,
                                            long startTime, String function) throws IOException, InterruptedException {
```

```
        // variables to save total red/green energy
        float importedEnergy = 0.0F, exportedEnergy = 0.0F, redEnergy = 0.0F, greenEnergy = 0.0F, comfortPercentage;
        int comfortCost = 0, matchComfort = 0;
        int currentMonth = 1;
        float[] importedEnergyMonthly = new float[12];
        float[] exportedEnergyMonthly = new float[12];
        float[] redEnergyMonthly = new float[12];
        float[] greenEnergyMonthly = new float[12];
        float[] selfconsumption = new float[12];
        for (int i = 0; i < 12; i++){
            importedEnergyMonthly[i] = 0.0F;
            exportedEnergyMonthly[i] = 0.0F;
            redEnergyMonthly[i] = 0.0F;
            greenEnergyMonthly[i] = 0.0F;
            selfconsumption[i] = 0.0F;
        }
        // date formatter to read from data
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");

        // for each sample of data
        for (String[] hour : totalConsumption) {

            LocalDateTime currentDateTime = LocalDateTime.parse(hour[0], formatter);
            if (currentDateTime.getMonthValue() != currentMonth){
                selfconsumption[currentMonth - 1] = greenEnergyMonthly[currentMonth - 1]
                        - exportedEnergyMonthly[currentMonth - 1];
                importedEnergyMonthly[currentMonth - 1] *= -1;
                ++currentMonth;
            }

            // read consumption and generation per hour from data
            float energyConsumption = 0.0F;
            for (int j = 1; j < hour.length - 1; j++) {
                float temp = Float.parseFloat(hour[j]);
                for (String[] rule : rules) {
                    if (rule[0].equals(hour[0]) && Integer.parseInt(rule[1]) == j) {
                        if (Float.parseFloat(hour[Integer.parseInt(rule[1])]) > 0.0) {
                            matchComfort += 1;
                        } else {
                            comfortCost += 1000;
```

```
                    }
                }
            }
            if (check == 0)
                energyConsumption += (temp * averageConsumption[j - 1]);
            else
                energyConsumption += (temp);
        }
        float energyGenerated = Float.parseFloat(hour[hour.length - 1]);

        // increase red/green energy
        redEnergy += energyConsumption;
        greenEnergy += energyGenerated;
        redEnergyMonthly[currentMonth - 1] += energyConsumption;
        greenEnergyMonthly[currentMonth - 1] += energyGenerated;

        // calculate generate minus consumption
        float typeOfEnergy = energyGenerated - energyConsumption;

        // greater than zero equals green
        if (typeOfEnergy >= 0.0) {
            exportedEnergy += typeOfEnergy;
            exportedEnergyMonthly[currentMonth - 1] += typeOfEnergy;
        }

        // less than zero equals red
        else {
            importedEnergy += typeOfEnergy;
            importedEnergyMonthly[currentMonth - 1] += typeOfEnergy;
        }
    }

    importedEnergyMonthly[11] *= -1;
    selfconsumption[11] = greenEnergyMonthly[11] - exportedEnergyMonthly[11];

    comfortPercentage = (float) (matchComfort * 100.0) / rules.size();

    float co2emissionPerKWh = (float) 0.449;
```

```
    // print total red/green energy
    System.out.printf("Total Exported Energy(Production energy non-consumed): %.2f kWh\n", exportedEnergy);
    System.out.println("Monthly Exported Energy:");
    for (int i = 0; i < 12; i++) {
        System.out.printf("%.2f, ", exportedEnergyMonthly[i]);
    }
    float selfConsumptionEnergy = greenEnergy - exportedEnergy;
    System.out.printf("\nTotal Self-Consumed Energy: %.2f kWh\n", selfConsumptionEnergy);
    System.out.println("Monthly Self-Consumed Energy");
    for (int i = 0; i < 12; i++) {
        System.out.printf("%.2f, ", selfconsumption[i]);
    }
    System.out.printf("\nTotal Imported Energy(Red energy): %.2f kWh\n", (-1.0 * importedEnergy));
    System.out.println("Monthly Imported Energy:");
    for (int i = 0; i < 12; i++) {
        System.out.printf("%.2f, ", importedEnergyMonthly[i]);
    }
    System.out.printf("\nTotal Production Energy(Exported + Self-consumed): %.2f kWh\n", greenEnergy);
    System.out.println("Monthly Production Energy:");
    for (int i = 0; i < 12; i++) {
        System.out.printf("%.2f, ", greenEnergyMonthly[i]);
    }
    System.out.printf("\nTotal Consumption Energy(Imported + Self-consumed): %.2f kWh\n", redEnergy);
    System.out.println("Monthly Consumption Energy:");
    for (int i = 0; i < 12; i++) {
        System.out.printf("%.2f, ", redEnergyMonthly[i]);
    }
    float co2emissions = -1 * importedEnergy * co2emissionPerKWh;
    System.out.printf("\nTotal CO2 emissions(USA avg: 0.449 kgCO2/kWh): %.2f kgCO2\n", co2emissions);
    System.out.printf("Total Comfort Error Cost: %d\n", comfortCost);
    System.out.printf("Comfort: %.2f %%\n", comfortPercentage);
    long endTime = System.nanoTime();
    float execTime = (float) ((endTime - startTime) / 1000000000.0);
    System.out.printf("That took %.2f seconds\n", execTime);

    // send results to server
//      ApiCalls.sendToAPI(function, exportedEnergy, selfConsumptionEnergy,
//              (float) (-1.0 * importedEnergy), greenEnergy, redEnergy,
//              co2emissions, comfortCost, comfortPercentage, execTime);
```

```
231     }
232
233 @   public static float[] applianceAverageConsumptionPerHour(List<String[]> energyConsumption) {
234
235         // init array
236         float[] averageConsumptions = new float[19];
237
238         // for each appliance calculate total consumption and how many times is greater than zero
239         for (int i = 0; i < 19; i++) {
240             int onCounter = 0;
241             float totalConsumption = 0.0F;
242
243             for (String[] row : energyConsumption) {
244                 if (Float.parseFloat(row[i + 1]) > 0.0) {
245                     totalConsumption += Float.parseFloat(row[i + 1]);
246                     onCounter++;
247                 }
248             }
249
250             // compute and save average of each appliance
251             averageConsumptions[i] = totalConsumption / onCounter;
252
253         }
254
255         return averageConsumptions;
256
257     }
258
259 @   public static float[] applianceMaxConsumption(List<String[]> energyConsumption) {
260         float[] maxConsumptions = new float[19];
261
262         for (int i = 1; i < energyConsumption.get(0).length - 1; i++) {
263             float maxConsTemp = 0.0F;
264             for (String[] hour : energyConsumption) {
265                 if (Float.parseFloat(hour[i]) > maxConsTemp) {
266                     maxConsTemp = Float.parseFloat(hour[i]);
267                 }
268             }
269             maxConsumptions[i - 1] = maxConsTemp;
270         }
271
272         return maxConsumptions;
273     }
274
275 }
```

```java
package cs.ucy.ac.cy;

import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.*;

public class Algorithms {

    // init random
    static Random rn = new Random();

    // function to calculate total import/export energy
    public static float calculateImportedExported(float[][] dayConsumption, float[] energyGenerated,
                                                  float[] averageConsumption, int check) {
        // variable to save total imported/exported energy of the day
        float totalImportedExported = 0.0F;
        for (int i = 0; i < dayConsumption.length; i++) {

            // calculate hour consumption
            float hourConsumption = 0.0F;
            for (int j = 0; j < dayConsumption[i].length; j++) {
                if (check == 1){
                    hourConsumption += (dayConsumption[i][j]);
                }
                else {
                    hourConsumption += (dayConsumption[i][j] * averageConsumption[j]);
                }
            }

            // add the imported/exported value to total of the day
            //if (hourConsumption > energyGenerated[i])
            totalImportedExported += Math.abs(energyGenerated[i] - hourConsumption);
        }

        // return the value
        return totalImportedExported;
    }
```

```java
    // function to random allocate hourly appliance consumption per day
    public static List<String[]> randomBinary(int iterations, List<String[]> applianceConsumption,
                                              float[] averageConsumption) throws IOException {

        // save appliance consumption to 2d array
        float[][] consumptions = new float[applianceConsumption.size()][19];
        for (int i = 0; i < applianceConsumption.size(); i++) {
            for (int j = 1; j < eppliance Consumption.get(i).length - 1; j++) {
                consumptions[i][j - 1] = Float.parseFloat(appliance Consumption.get(i)[j]);
            }
        }

        // list for writing data (for output)
        List<float[]> energyConsumptions = new ArrayList<>();

        // iterate day per day (every 24 hours)
        for (int i = 0; i < consumptions.length; i += 24) {

            // get the daily energy generation
            float[] energyGenerated = new float[24];
            for (int j = 0; j < energyGenerated.length; j++) {
                energyGenerated[j] = Float.parseFloat(applianceConsumption.get(j + i)[20]);
            }

            // get the daily energy consumption per appliance (used for random operation) and set best as the current
            float[][] dayHourlyConsumption = new float[24][19];
            float[][] bestDayHourlyConsumption = new float[24][19];
            for (int j = i; j < i + 24; j++) {
                for (int k = 0; k < 19; k++) {
                    dayHourlyConsumption[j % 24][k] = consumptions[j][k];
                    bestDayHourlyConsumption[j % 24][k] = consumptions[j][k];
                }
            }

            // calculate current total import and exported energy
            float bestImportedExported = calculateImportedExported(dayHourlyConsumption,
                    energyGenerated, averageConsumption, check: 0);
```

```
        // for each day random allocate n times and get the lowest import/export
        for (int k = 0; k < iterations; k++) {

            // set an array with 0 (used to random allocate appliance consumption)
            dayHourlyConsumption = new float[24][19];
            for (float[] floats : dayHourlyConsumption) {
                Arrays.fill(floats, val: 0.0F);
            }

            // counter to count 24 hours of the day
            int counter = 0;

            // iterate above every hour of the day
            for (int j = i; j < i + 24; j++) {

                // hourly random allocation of devices in a day
                for (int l = 0; l < 19; l++) {
                    if (bestDayHourlyConsumption[j % 24][l] != 0) {
                        int rnum = rn.nextInt( bound: 24);
                        if (dayHourlyConsumption[rnum][l] == 0) {
                            dayHourlyConsumption[rnum][l] = bestDayHourlyConsumption[j % 24][l];
                            continue;
                        }
                        for (; ; ) {
                            rnum = rn.nextInt( bound: 24);
                            if (dayHourlyConsumption[rnum][l] == 0) {
                                dayHourlyConsumption[rnum][l] = bestDayHourlyConsumption[j % 24][l];
                                break;
                            }
                        }
                    }
                }

                // if hour = 23 (last hour of the day) write results in list
                if (counter == 23) {

                    // calculate total imported and exported energy
                    float totalImportedExported = calculateImportedExported(dayHourlyConsumption,
                            energyGenerated, averageConsumption, check: 0);

                    // if total import/export is less than the previous save the day hourly energy consumption
                    if (totalImportedExported < bestImportedExported) {

                    // save the new appliance consumption per hour
                    bestDayHourlyConsumption = new float[24][19];
                    for (int l = 0; l < dayHourlyConsumption.length; l++) {
                        System.arraycopy(dayHourlyConsumption[l], srcPos: 0, bestDayHourlyConsumption[l],
                                destPos: 0, dayHourlyConsumption[l].length);
                    }

                    //set the new import/export energy
                    bestImportedExported = totalImportedExported;
                    }
                }
                // increase the counter
                else {
                    counter++;
                }
            }
        }

        // add 24 results (one per hour) to list
        energyConsumptions.addAll(Arrays.asList(bestDayHourlyConsumption));
    }

    List<String[]> output = new ArrayList<>();
    for (int i = 0; i < energyConsumptions.size(); i++) {
        String[] array = new String[21];
        array[0] = applianceConsumption.get(i)[0];
        for (int j = 0; j < energyConsumptions.get(i).length; j++) {
            array[j + 1] = String.valueOf(energyConsumptions.get(i)[j]);
        }
        array[20] = applianceConsumption.get(i)[20];
        output.add(array);
    }

    // write results in csv file
    File randomTotalFile = new File( pathname: "RandomBinaryTotalConsumptionPerHour.csv");
    FileWriter totalWriter = new FileWriter(randomTotalFile);
    for (int i = 0; i < energyConsumptions.size(); i++) {
        float[] row = energyConsumptions.get(i);
```

```
            float total = 0.0F;
            for (int j = 0; j < row.length; j++) {
                total += (row[j] * averageConsumption[j]);
            }
            totalWriter.write( str: applianceConsumption.get(i)[0] + "," + total +
                    "," + applianceConsumption.get(i)[20] + "\n");
        }
        totalWriter.close();

        return output;

    }

    public static List<String[]> random(int iterations, List<String[]> applianceConsumption,
                                        float[] maxConsumption, float[] averageConsumption) throws IOException {

        // save appliance consumption to 2d array
        float[][] consumptions = new float[applianceConsumption.size()][19];
        for (int i = 0; i < applianceConsumption.size(); i++) {
            for (int j = 1; j < applianceConsumption.get(i).length - 1; j++) {
                consumptions[i][j - 1] = Float.parseFloat(applianceConsumption.get(i)[j]);
            }
        }

        // list for writing data (for output)
        List<float[]> energyConsumptions = new ArrayList<>();

        // iterate day per day (every 24 hours)
        for (int i = 0; i < consumptions.length; i += 24) {

            // get the daily energy generation
            float[] energyGenerated = new float[24];
            for (int j = 0; j < energyGenerated.length; j++) {
                energyGenerated[j] = Float.parseFloat(applianceConsumption.get(j + i)[20]);
            }


            // get the daily energy consumption per appliance (used for random operation) and set best as the current
            float[][] dayHourlyConsumption = new float[24][19];
            float[][] bestDayHourlyConsumption = new float[24][19];
            for (int j = i; j < i + 24; j++) {
                for (int k = 0; k < 19; k++) {
                    dayHourlyConsumption[j % 24][k] = consumptions[j][k];
                    bestDayHourlyConsumption[j % 24][k] = consumptions[j][k];
                }
            }

            // calculate current total import and exported energy
            float bestImportedExported = calculateImportedExported(dayHourlyConsumption,
                    energyGenerated, averageConsumption, check: 1);

            // for each day random allocate n times and get the lowest import/export
            for (int k = 0; k < iterations; k++) {

                // set an array with 0 (used to random allocate appliance consumption)
                dayHourlyConsumption = new float[24][19];
                for (float[] floats : dayHourlyConsumption) {
                    Arrays.fill(floats, val: 0.0F);
                }

                // counter to count 24 hours of the day
                int counter = 0;

                // iterate above every hour of the day
                for (int j = i; j < i + 24; j++) {

                    // hourly random allocation of devices in a day
                    for (int l = 0; l < 19; l++) {
                        if (bestDayHourlyConsumption[j % 24][l] != 0) {
                            int rnum = rn.nextInt( bound: 24);
                            if (dayHourlyConsumption[rnum][l] + bestDayHourlyConsumption[j % 24][l]
                                    <= maxConsumption[l]){
                                dayHourlyConsumption[rnum][l] += bestDayHourlyConsumption[j % 24][l];
                            }
                            else{
                                for (;;){
```

```
                        rnum = rn.nextInt( bound: 24);
                        if (dayHourlyConsumption[rnum][l] + bestDayHourlyConsumption[j % 24][l]
                                <= maxConsumption[l]){
                            dayHourlyConsumption[rnum][l] += bestDayHourlyConsumption[j % 24][l];
                            break;
                        }
                    }
                }


            }
        }

        // if hour = 23 (last hour of the day) write results in list
        if (counter == 23) {

            // calculate total imported and exported energy
            float totalImportedExported = calculateImportedExported(dayHourlyConsumption,
                    energyGenerated, averageConsumption, check: 1);

            // if total import/export is less than the previous save the day hourly energy consumption
            if (totalImportedExported < bestImportedExported) {

            // save the new appliance consumption per hour
            bestDayHourlyConsumption = new float[24][19];
            for (int l = 0; l < dayHourlyConsumption.length; l++) {
                System.arraycopy(dayHourlyConsumption[l], srcPos: 0, bestDayHourlyConsumption[l],
                        destPos: 0, dayHourlyConsumption[l].length);
            }

            // set the new import/export energy
            bestImportedExported = totalImportedExported;
            }
        }
        // increase the counter
        else {
            counter++;
        }
    }
```

```
        }

        // add 24 results (one per hour) to list
        energyConsumptions.addAll(Arrays.asList(bestDayHourlyConsumption));
    }

    List<String[]> output = new ArrayList<>();
    for (int i = 0; i < energyConsumptions.size(); i++) {
        String[] array = new String[21];
        array[0] = applianceConsumption.get(i)[0];
        for (int j = 0; j < energyConsumptions.get(i).length; j++) {
            array[j + 1] = String.valueOf(energyConsumptions.get(i)[j]);
        }
        array[20] = applianceConsumption.get(i)[20];
        output.add(array);
    }

    // write results in csv file
    File randomTotalFile = new File( pathname: "RandomTotalConsumptionPerHour.csv");
    FileWriter totalWriter = new FileWriter(randomTotalFile);
    for (int i = 0; i < energyConsumptions.size(); i++) {
        float[] row = energyConsumptions.get(i);
        float total = 0.0F;
        for (int j = 0; j < row.length; j++) {
            total += (row[j] * averageConsumption[j]);
        }
        totalWriter.write( str: applianceConsumption.get(i)[0] + "," + total +
                "," + applianceConsumption.get(i)[20] + "\n");
    }
    totalWriter.close();

    return output;

}
```

```java
    // function to create on/off dataset if energy consumption > 0
    public static void convertToBinary(ImportedData data) {

        List<String[]> hourlyEnergyConsumptions = data.getHourlyEnergyConsumption();

        List<String[]> hourlyBinaryData = new ArrayList<>();

        for (String[] hourlyEnergyConsumption : hourlyEnergyConsumptions) {
            String[] binaryHourConsumption = new String[21];
            binaryHourConsumption[0] = hourlyEnergyConsumption[0];
            for (int j = 1; j < hourlyEnergyConsumption.length - 1; j++) {
                if (!hourlyEnergyConsumption[j].equals("0.0")) {
                    binaryHourConsumption[j] = "1.0";
                } else {
                    binaryHourConsumption[j] = "0.0";
                }
            }
            binaryHourConsumption[20] = hourlyEnergyConsumption[20];
            hourlyBinaryData.add(binaryHourConsumption);
        }

        data.setBinaryEnergyConsumption(hourlyBinaryData);

    }

    public static List<String[]> bruteForceBinarySolution(List<String[]> applianceConsumption,
                                                          float[] averageConsumption) throws IOException {

        List<String[]> solution = new ArrayList<>();

        for (int i = 0; i < applianceConsumption.size(); i += 24) {

            // copy the consumptions of the day
            String[][] day = new String[24][21];
            String[][] tempDay = new String[24][21];
            for (int j = 0; j < day.length; j++) {
                for (int k = 0; k < day[j].length; k++) {
                    day[j][k] = applianceConsumption.get(j + i)[k];
                    tempDay[j][k] = applianceConsumption.get(j + i)[k];
                }
            }

            // brute force reallocate to find optimal solution
            for (int k = 1; k < day[0].length - 1; k++) {
                for (int pos = 0; pos < day.length; pos++) {
                    for (int j = 1; j < day.length - pos - 1; j++) {
                        String tmp = tempDay[pos][k];
                        tempDay[pos][k] = tempDay[pos + j][k];
                        tempDay[pos + j][k] = tmp;

                        if (checkRedGreenBinary(day, tempDay, averageConsumption)) {
                            String temp = day[pos][k];
                            day[pos][k] = day[pos + j][k];
                            day[pos + j][k] = temp;
                        } else {
                            String tmp1 = tempDay[pos + j][k];
                            tempDay[pos + j][k] = tempDay[pos][k];
                            tempDay[pos][k] = tmp1;
                        }
                    }
                }

                for (int pos = day.length - 1; pos > 0; pos--) {
                    for (int j = 1; j < pos - 1; j++) {
                        String tmp = tempDay[pos][k];
                        tempDay[pos][k] = tempDay[pos - j][k];
                        tempDay[pos - j][k] = tmp;

                        if (checkRedGreenBinary(day, tempDay, averageConsumption)) {
                            String temp = day[pos][k];
                            day[pos][k] = day[pos - j][k];
                            day[pos - j][k] = temp;
                        } else {
                            String tmp1 = tempDay[pos - j][k];
                            tempDay[pos - j][k] = tempDay[pos][k];
                            tempDay[pos][k] = tmp1;
                        }
                    }
                }
```

```
        }

        for (int k = day[0].length - 1; k > 0; k--) {
            for (int pos = 0; pos < day.length; pos++) {
                for (int j = 1; j < day.length - pos - 1; j++) {
                    String tmp = tempDay[pos][k];
                    tempDay[pos][k] = tempDay[pos + j][k];
                    tempDay[pos + j][k] = tmp;

                    if (checkRedGreenBinary(day, tempDay, averageConsumption)) {
                        String temp = day[pos][k];
                        day[pos][k] = day[pos + j][k];
                        day[pos + j][k] = temp;
                    } else {
                        String tmp1 = tempDay[pos + j][k];
                        tempDay[pos + j][k] = tempDay[pos][k];
                        tempDay[pos][k] = tmp1;
                    }
                }
            }

            for (int pos = day.length - 1; pos > 0; pos--) {
                for (int j = 1; j < pos - 1; j++) {
                    String tmp = tempDay[pos][k];
                    tempDay[pos][k] = tempDay[pos - j][k];
                    tempDay[pos - j][k] = tmp;

                    if (checkRedGreenBinary(day, tempDay, averageConsumption)) {
                        String temp = day[pos][k];
                        day[pos][k] = day[pos - j][k];
                        day[pos - j][k] = temp;
                    } else {
                        String tmp1 = tempDay[pos - j][k];
                        tempDay[pos - j][k] = tempDay[pos][k];
                        tempDay[pos][k] = tmp1;
                    }
                }
            }

        }
```

```
            // add results to solution list
            Collections.addAll(solution, day);
        }

        File bruteTotalFile = new File( pathname: "BruteForceBinaryTotalConsumptionPerHour.csv");
        FileWriter bruteWriter = new FileWriter(bruteTotalFile);
        for (String[] row : solution) {

            for (int j = 0; j < row.length - 1; j++) {
                bruteWriter.write( str: row[j] + ",");
            }
            bruteWriter.write( str: row[20] + "\n");
        }
        bruteWriter.close();

        return solution;
    }

    // function to calculate total import/export energy
    public static boolean checkRedGreenBinary(String[][] day, String[][] tempDay, float[] averageConsumption) {
        // variable to save total imported/exported energy of the day
        float totalDayImportedExported = 0.0F, totalTempDayImportedExported = 0.0F;
        for (int i = 0; i < day.length; i++) {

            // calculate hour consumption
            float hourDay = 0.0F, hourTempDay = 0.0F;
            for (int j = 1; j < day[i].length - 1; j++) {
                hourDay += (Float.parseFloat(day[i][j]) * averageConsumption[j - 1]);
                hourTempDay += (Float.parseFloat(tempDay[i][j]) * averageConsumption[j - 1]);
            }

            // add the imported/exported value to total of the day
            totalDayImportedExported += Math.abs(Float.parseFloat(day[i][20]) - hourDay);
            totalTempDayImportedExported += Math.abs(Float.parseFloat(tempDay[i][20]) - hourTempDay);
        }
```

```
        return totalTempDayImportedExported < totalDayImportedExported;
    }

    public static List<String[]> bruteForceSolution(List<String[]> applianceConsumption,
                                                    float[] maxConsumption) throws IOException {

        List<String[]> solution = new ArrayList<>();

        for (int i = 0; i < applianceConsumption.size(); i += 24) {

            // copy the consumptions of the day
            String[][] day = new String[24][21];
            for (int j = 0; j < day.length; j++) {
                System.arraycopy(applianceConsumption.get(j + i), srcPos: 0, day[j], destPos: 0, day[j].length);
            }

            Stack<String[][]> stack = new Stack<>();
            stack.push(day);

            // brute force reallocate to find optimal solution
            for (int k = 1; k < day[0].length - 1; k++) {

                for (int pos = 0; pos < day.length; pos++) {
                    String[][] check = new String[24][21];
                    for (int c = 0; c < check.length; c++) {
                        check[c] = stack.peek()[c].clone();
                    }
                    String tmp = check[pos][k];
                    check[pos][k] = "0.0";
                    for (int j = 1; j < day.length - pos - 1; j++) {
                        String[][] checkTemp = new String[24][21];
                        for (int c = 0; c < checkTemp.length; c++) {
                            checkTemp[c] = check[c].clone();
                        }
                        if ((Float.parseFloat(checkTemp[pos + j][k]) + Float.parseFloat(tmp))
                                <= maxConsumption[k - 1]) {
                            checkTemp[pos + j][k] = String.valueOf((Float.parseFloat(checkTemp[pos + j][k])
                                    + Float.parseFloat(tmp)));

                            if (checkRedGreen(stack.peek(), checkTemp)) {
                                stack.push(checkTemp);
                            }
                        }
                    }
                }

                for (int pos = day.length - 1; pos > 0; pos--) {
                    String[][] check = new String[24][21];
                    for (int c = 0; c < check.length; c++) {
                        check[c] = stack.peek()[c].clone();
                    }
                    String tmp = check[pos][k];
                    check[pos][k] = "0.0";
                    for (int j = 1; j < pos - 1; j++) {
                        String[][] checkTemp = new String[24][21];
                        for (int c = 0; c < checkTemp.length; c++) {
                            checkTemp[c] = check[c].clone();
                        }
                        if ((Float.parseFloat(checkTemp[pos - j][k]) + Float.parseFloat(tmp))
                                <= maxConsumption[k - 1]) {
                            checkTemp[pos - j][k] = String.valueOf((Float.parseFloat(checkTemp[pos - j][k])
                                    + Float.parseFloat(tmp)));
                            if (checkRedGreen(stack.peek(), checkTemp)) {
                                stack.push(checkTemp);
                            }
                        }
                    }
                }
            }



            String[][] daySolution = stack.peek();

            // add results to solution list
            Collections.addAll(solution, daySolution);
        }

```

```
        File bruteTotalFile = new File( pathname: "BruteForceTotalConsumptionPerHour.csv");
        //bruteTotalFile.createNewFile();
        FileWriter bruteWriter = new FileWriter(bruteTotalFile);
        for (String[] row : solution) {

            for (int j = 0; j < row.length - 1; j++) {
                bruteWriter.write( str: row[j] + ",");
            }
            bruteWriter.write( str: row[20] + "\n");
        }
        bruteWriter.close();

        return solution;
    }

    // function to calculate total import/export energy
    public static boolean checkRedGreen(String[][] day, String[][] tempDay) {
        // variable to save total imported/exported energy of the day
        float totalImpExp = 0.0F, totalTempImpExp = 0.0F;
        for (int i = 0; i < day.length; i++) {

            // calculate hour consumption
            float hourConsumption = 0.0F, hourTempConsumption = 0.0F;
            for (int j = 1; j < day[i].length -1; j++) {
                hourConsumption += Float.parseFloat(day[i][j]);
                hourTempConsumption += Float.parseFloat(tempDay[i][j]);
            }

            // add the imported/exported value to total of the day
//            if (hourTempConsumption > energyGenerated[i])
            totalImpExp += Math.abs(Float.parseFloat(day[i][20]) - hourConsumption);
            totalTempImpExp += Math.abs(Float.parseFloat(day[i][20]) - hourTempConsumption);
        }

        // return the value
        return totalTempImpExp < totalImpExp;
    }
}
```

```
package cs.ucy.ac.cy;

import org.json.JSONArray;
import org.json.JSONObject;

import java.io.IOException;
import java.net.URI;
import java.net.http.HttpClient;
import java.net.http.HttpRequest;
import java.net.http.HttpResponse;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.ArrayList;
import java.util.List;

public class ApiCalls {

    public static void sendToAPI(String function, float exportedEnergy, float selfConsumptionEnergy,
                                 float importedEnergy, float production, float consumption,
                                 float co2emissions, float comfortCost, float comfortPercentage,
                                 float execTime) throws IOException, InterruptedException {
        JSONObject jsonObject = new JSONObject()
                .put("function_name", String.valueOf(function))
                .put("exported", String.valueOf(exportedEnergy))
                .put("selfconsumption", String.valueOf(selfConsumptionEnergy))
                .put("imported", String.valueOf(importedEnergy))
                .put("production", String.valueOf(production))
                .put("consumption", String.valueOf(consumption))
                .put("co2_emissions", String.valueOf(co2emissions))
                .put("comfort_error", String.valueOf(comfortCost))
                .put("comfort", String.valueOf(comfortPercentage))
                .put("execution_time", String.valueOf(execTime));

        System.out.println(jsonObject.toString());


        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create("http://10.16.30.215/api/update-results"))
                .header( name: "Content-Type",  value: "application/json")
                .POST(HttpRequest.BodyPublishers.ofString(jsonObject.toString()))
```

```
        HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());
        System.out.println(response.statusCode() + " - Results send to server");
        System.out.println(response.body());
    }

    public static List<String[]> getRulesFromAPI() throws IOException, InterruptedException {
        HttpClient client = HttpClient.newHttpClient();
        HttpRequest request = HttpRequest.newBuilder()
                .uri(URI.create("http://10.16.30.215/api/get-metarules"))
                .GET()
                .build();

        HttpResponse<String> response = client.send(request, HttpResponse.BodyHandlers.ofString());

        System.out.println(response.statusCode() + " - Rules received from server");

        List<String[]> rules = new ArrayList<>();
        JSONArray jsonArray = new JSONArray(response.body());

        for (int i = 0; i < jsonArray.length(); i++) {
            String[] rule = new String[2];
            DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm");
            DateTimeFormatter formatter2 = DateTimeFormatter.ofPattern("dd/MM/yyyy HH:mm");
            LocalDateTime dateTime = LocalDateTime.parse(jsonArray.getJSONObject(i).getString( key: "time"), formatter);
            rule[0] = dateTime.format(formatter2);
            rule[1] = jsonArray.getJSONObject(i).getString( key: "action");

            rules.add(rule);
        }

        return rules;
    }
}
```

```
package cs.ucy.ac.cy;

import java.io.IOException;
import java.util.*;

public class Chromosome implements Cloneable {

    // object parameters init
    private float fitness;
    private float exportedEnergy;
    private float importedEnergy;
    private float totalConsumption;
    private float totalProduction;
    private List<String[]> genes;

    // chromosome constructor with the consumptions
    public Chromosome(List<String[]> energyConsumptions, float[] maxConsumption,
                      float[] averageConsumption) throws IOException {

        // set genes using random allocation of appliances
        this.genes = Algorithms.random( iterations: 1, energyConsumptions, maxConsumption, averageConsumption);
        this.fitness = 0.0F;
        this.exportedEnergy = 0.0F;
        this.importedEnergy = 0.0F;
        this.totalConsumption = 0.0F;
        this.totalProduction = 0.0F;

    }

    public Chromosome(List<String[]> list) {
        this.genes = list;
        this.fitness = 0.0F;
        this.exportedEnergy = 0.0F;
        this.importedEnergy = 0.0F;
        this.totalConsumption = 0.0F;
        this.totalProduction = 0.0F;
    }

```

```java
    // chromosome constructor with chromosome
    public Chromosome(Chromosome c) {
        this.genes = c.getGenes();
        this.fitness = c.getFitness();
        this.exportedEnergy = c.getExportedEnergy();
        this.importedEnergy = c.getImportedEnergy();
        this.totalConsumption = c.getTotalConsumption();
        this.totalProduction = c.getTotalProduction();
    }

    // calculate fitness
    public void calcFitness() {

        float totalImported = 0.0F;
        float totalExported = 0.0F;
        float totalConsumption = 0.0F;
        float totalProduction = 0.0F;

        for (String[] gene : this.genes) {

            // calculate hour consumption
            float hourConsumption = 0.0F;
            for (int j = 0; j < gene.length - 2; j++) {
                hourConsumption += (Float.parseFloat(gene[j + 1]));
            }

            totalConsumption += hourConsumption;
            totalProduction += Float.parseFloat(gene[20]);

            // if consumption is greater than production energy equals imported
            if (hourConsumption > Float.parseFloat(gene[20]))
                totalImported += (hourConsumption - Float.parseFloat(gene[20]));
            else
                totalExported += (Float.parseFloat(gene[20]) - hourConsumption);
        }

        // set values
        this.fitness = totalImported;
        this.importedEnergy = totalImported;
```

```java
        this.exportedEnergy = totalExported;
        this.totalConsumption = totalConsumption;
        this.totalProduction = totalProduction;
    }

    public void fixConsumptions(float[][] dailyConsumptions, float[] maxConsumption) {

        // threshold to check if two floats are equal
        final float THRESHOLD = (float) .00001;

        List<String[]> output = new ArrayList<>();

        // iterate day per day
        for (int i = 0; i < this.genes.size(); i += 24) {

            // copy in array the current day
            String[][] day = new String[24][21];
            for (int j = i; j < i + 24; j++) {
                System.arraycopy(this.genes.get(j), srcPos: 0, day[j % 24], destPos: 0, length: 21);
            }

            // find how many times devices are open in the day after gen. alg.
            float[] checkConsumptions = new float[19];
            for (int j = 1; j < this.genes.get(i).length - 1; j++) {
                float counter = 0;
                for (int k = 0; k < 24; k++) {
                    counter += Float.parseFloat(day[k][j]);
                }
                checkConsumptions[j - 1] = counter;
            }

            Integer[] idx = new Integer[24];
            for (int j = 0; j < 24; j++) {
                idx[j] = j;
            }
            float[] productions = new float[24];
            for (int j = 0; j < 24; j++) {
                productions[j] = Float.parseFloat(day[j][20]);
            }
```

```
        Arrays.sort(idx, (o1, o2) -> Float.compare(productions[o1], productions[o2]));

        // fix the consumption by iterate device per device
        for (int j = 0; j < checkConsumptions.length; j++) {

            float consumptionDiff;

            if (Math.abs(checkConsumptions[j] - dailyConsumptions[i / 24][j]) < THRESHOLD) continue;
            // if is greater than the original turn off devices
            else if (checkConsumptions[j] > dailyConsumptions[i / 24][j]) {
                consumptionDiff = checkConsumptions[j] - dailyConsumptions[i / 24][j];
                for (Integer integer : idx) {

                    if (day[integer][j + 1].equals("0.0")) continue;
                    else {

                        if (consumptionDiff <= 0)
                            break;

                        float currCons = Float.parseFloat(day[integer][j + 1]);
                        if (currCons <= consumptionDiff) {
                            this.genes.get(i + integer)[j + 1] = "0.0";
                            consumptionDiff -= currCons;
                        } else {
                            float tmp = currCons - consumptionDiff;
                            day[integer][j + 1] = String.valueOf(tmp);
                            break;
                        }
                    }
                }
            }

            // if is less than the original turn on devices
            else if (checkConsumptions[j] < dailyConsumptions[i / 24][j]) {
                consumptionDiff = dailyConsumptions[i / 24][j] - checkConsumptions[j];
                for (int l = idx.length - 1; l >= 0; l--) {
```

```
                    if (consumptionDiff <= 0)
                        break;

                    float currCons = Float.parseFloat(day[idx[l]][j + 1]);
                    if ((currCons + consumptionDiff) >= maxConsumption[j]) {
                        day[idx[l]][j + 1] = String.valueOf(maxConsumption[j]);
                        consumptionDiff -= (maxConsumption[j] - currCons);
                    } else {
                        float tmp = currCons + consumptionDiff;
                        day[idx[l]][j + 1] = String.valueOf(tmp);
                        break;
                    }

                }
            }
        }

        Collections.addAll(output, day);
    }

    this.genes = new ArrayList<>(output);

}

@Override
protected Object clone() throws CloneNotSupportedException {
    Chromosome chromosome = (Chromosome) super.clone();
    chromosome.genes = new ArrayList<>();
    chromosome.genes.addAll(this.genes);
    chromosome.fitness = this.fitness;
    chromosome.importedEnergy = this.importedEnergy;
    chromosome.exportedEnergy = this.exportedEnergy;
    chromosome.totalConsumption = this.totalConsumption;
    chromosome.totalProduction = this.totalProduction;
    return chromosome;
}

// fitness getter
public float getFitness() { return fitness; }
```

```
    // exported energy getter
    public float getExportedEnergy() { return exportedEnergy; }

    // imported energy getter
    public float getImportedEnergy() { return importedEnergy; }

    // genes getter
    public List<String[]> getGenes() { return genes; }

    // total consumption getter
    public float getTotalConsumption() { return totalConsumption; }

    // total production getter
    public float getTotalProduction() { return totalProduction; }

}
```

```
package cs.ucy.ac.cy;

import java.io.IOException;
import java.util.List;

public class Population {

    // object parameters init
    private final int popSize;
    private Chromosome[] chromosomes;
    private float fittestScore = 0.0F;
    private float fittestExported = 0.0F;
    private float fittestImported = 0.0F;
    private float fittestConsumption = 0.0F;
    private float fittestProduction = 0.0F;
    private int fittestIndex = -1;

    // population object constructor
    public Population(int popSize, List<String[]> energyConsumptions, float[] maxConsumption,
                      float[] averageConsumption) throws IOException {
        super();
        this.popSize = popSize;
        this.chromosomes = new Chromosome[popSize];

        // create a first population pool
        for (int i = 0; i < popSize; i++) {
            chromosomes[i] = new Chromosome(energyConsumptions, maxConsumption, averageConsumption);
        }
    }

    // get the fittest individual (individual with the lowest fitness)
    public void getFittest() {
        float maxFit = Float.MAX_VALUE;
        int maxFitIndex = 0;
        for (int i = 0; i < chromosomes.length; i++) {
            if (chromosomes[i].getFitness() < maxFit) {
                maxFit = chromosomes[i].getFitness();
                maxFitIndex = i;
            }
        }
        fittestScore = chromosomes[maxFitIndex].getFitness();
```

```
        fittestImported = chromosomes[maxFitIndex].getImportedEnergy();
        fittestExported = chromosomes[maxFitIndex].getExportedEnergy();
        fittestConsumption = chromosomes[maxFitIndex].getTotalConsumption();
        fittestProduction = chromosomes[maxFitIndex].getTotalProduction();
        fittestIndex = maxFitIndex;

    }

    // get index of the least fit individual
    public int getLeastFittestIndex() {
        float minFitVal = Float.MIN_VALUE;
        int minFitIndex = 0;
        for (int i = 0; i < chromosomes.length; i++) {
            if (chromosomes[i].getFitness() > minFitVal) {
                minFitVal = chromosomes[i].getFitness();
                minFitIndex = i;
            }
        }
        return minFitIndex;
    }

    // calculate fitness of each individual
    public void calculateFitness() {

        for (Chromosome chromosome : chromosomes) {
            chromosome.calcFitness();
        }
        getFittest();
    }

    public int getPopSize() { return popSize; }

    // individuals getter
    public Chromosome[] getChromosomes() { return chromosomes; }

    // fittest score getter
    public float getFittestScore() { return fittestScore; }

    public float getFittestExported() { return fittestExported; }


    public float getFittestImported() { return fittestImported; }

    public float getFittestConsumption() { return fittestConsumption; }

    public float getFittestProduction() { return fittestProduction; }

    public int getFittestIndex() { return fittestIndex; }
}
```

```java
package cs.ucy.ac.cy;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Random;

public class GreenCap {

    // object parameters init
    private final Population population;
    private Chromosome firstSelected;
    private Chromosome secondSelected;
    private Chromosome best;
    private int generationCount;
    Random rn = new Random();

    // GreenCap object constructor
    public GreenCap(List<String[]> energyConsumptions, int popSize, float[] maxConsumption, float[] averageConsumption)
            throws IOException {
        this.population = new Population(popSize, energyConsumptions, maxConsumption, averageConsumption);
        this.generationCount = 0;
    }

    // function to implement allocation of appliances using genetic algorithm
    public static List<String[]> GeneticAlgorithm(List<String[]> energyConsumptions, List<String[]> peakHours,
                                                  float[] maxConsumption, float[] averageConsumption)
            throws IOException {

        // parameters init
        final int popSize = 50;
        final int maxGeneration = 10;

        // init GreenCap object - Population
        GreenCap greenCap = new GreenCap(energyConsumptions, popSize, maxConsumption, averageConsumption);

        // calculate how total consumption of a device in a day for all days of the year
        float[][] dailyConsumption = greenCap.calcConsumptions(energyConsumptions);


        //System.out.println("\nGreenCap:");
        System.out.println("Population of " + greenCap.population.getPopSize() + " chromosome(s).");

        //Calculate fitness of each individual
        greenCap.population.calculateFitness();

        // print current generation and fitness
        System.out.println("Generation: " + greenCap.generationCount + " Fittest Score: " +
                greenCap.population.getFittestScore() + " Exported Energy: " +
                greenCap.population.getFittestExported() + " kWh. Imported Energy: " +
                greenCap.population.getFittestImported() + " kWh. Total Production: " +
                greenCap.population.getFittestProduction() + " kWh. Total Consumption: " +
                greenCap.population.getFittestConsumption() + " kWh.");

        // while current generation is less than max number of generations
        while (greenCap.generationCount < maxGeneration) {

            int iteration = 0;

            // increase the current generation
            ++greenCap.generationCount;

            // while iteration is less than the number of individuals in population
            while (iteration < popSize) {

                // do selection (random select two individuals)
                greenCap.selection();

                // do crossover function
                greenCap.crossover();

                // do mutation function
                greenCap.mutation();

                // fix the consumptions to match the first consumption
                greenCap.firstSelected.fixConsumptions(dailyConsumption, maxConsumption);
                greenCap.secondSelected.fixConsumptions(dailyConsumption, maxConsumption);

                // update fitness values of offspring and clone best offspring
                greenCap.firstSelected.calcFitness();
```

```
            greenCap.secondSelected.calcFitness();
            try {
                if (greenCap.firstSelected.getFitness() < greenCap.secondSelected.getFitness())
                    greenCap.best = (Chromosome) greenCap.firstSelected.clone();
                else
                    greenCap.best = (Chromosome) greenCap.secondSelected.clone();
            } catch (CloneNotSupportedException e) {
                e.printStackTrace();
            }

            // reallocate appliances from peak to non-peak hour
            greenCap.peakHoursReallocation(peakHours);

            // add the fittest offspring to population
            greenCap.addFittestOffspring();

            // increase iteration
            ++iteration;
        }

        // find the fittest chromosome
        greenCap.population.getFittest();

        // print current generation and fitness
        System.out.println("Generation: " + greenCap.generationCount + " Fittest Score: " +
                greenCap.population.getFittestScore() + " Exported Energy: " +
                greenCap.population.getFittestExported() + " kWh. Imported Energy: " +
                greenCap.population.getFittestImported() + " kWh. Total Production: " +
                greenCap.population.getFittestProduction() + " kWh. Total Consumption: " +
                greenCap.population.getFittestConsumption() + " kWh.");

    }

    //System.out.println("Fitness: " + greenCap.population.getFittestScore());

    for (Chromosome c : greenCap.population.getChromosomes())
        c.fixConsumptions(dailyConsumption, maxConsumption);

    greenCap.population.calculateFitness();
```

```
    greenCap.population.getFittest();

    return greenCap.population.getChromosomes()[greenCap.population.getFittestIndex()].getGenes();

}

// selection function
public void selection() {

    try {
        // random select the first individual
        int firstRandom = rn.nextInt(population.getChromosomes().length);
        firstSelected = (Chromosome) population.getChromosomes()[firstRandom].clone();

        // random select the second individual (make sure second != first)
        int secondRandom = rn.nextInt(population.getChromosomes().length);
        while (secondRandom == firstRandom) {
            secondRandom = rn.nextInt(population.getChromosomes().length);
        }
        secondSelected = (Chromosome) population.getChromosomes()[secondRandom].clone();
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }

}

// crossover function
public void crossover() {

    //int crossOverPoint = rn.nextInt(24);

    // lists to save the results for crossover
    List<String[]> first = new ArrayList<>();
    List<String[]> second = new ArrayList<>();

    // iterate day per day
    for (int i = 0; i < firstSelected.getGenes().size(); i += 24) {

```

```
        // with possibility 90% swap values between the two offsprings at crossover point
        if (rn.nextInt( bound: 100) <= 90) {

            // select a random crossover point in the day
            int crossOverPoint = rn.nextInt( bound: 24);

            // swap consumption until the crossover point between the two offsprings
            for (int k = i; k < i + crossOverPoint; k++) {
                first.add(secondSelected.getGenes().get(k));
                second.add(firstSelected.getGenes().get(k));
            }

            // fill the list with the remaining hours of the day
            for (int k = i + crossOverPoint; k < i + 24; k++) {
                first.add(firstSelected.getGenes().get(k));
                second.add(secondSelected.getGenes().get(k));
            }
        }

        // else just fill with the current consumptions
        else {
            for (int k = i; k < i + 24; k++) {
                first.add(firstSelected.getGenes().get(k));
                second.add(secondSelected.getGenes().get(k));
            }
        }
    }

    // save results
    firstSelected = new Chromosome(first);
    secondSelected = new Chromosome(second);

}

// mutation function
public void mutation() {

    // select a random mutation point in the day for the first selected
    //int mutationPoint = rn.nextInt(24);

```

```
    // lists to save the results for mutation
    List<String[]> first = new ArrayList<>();
    List<String[]> second = new ArrayList<>();

    //iterate day per day
    for (int i = 0; i < firstSelected.getGenes().size(); i += 24) {

        // with possibility 1% turn on/off appliances at the first selected offspring at mutation point
        if (rn.nextInt( bound: 100) == 99) {

            // select a random mutation point in the day for the first selected
            int mutationPoint = rn.nextInt( bound: 24);

            // add to list until the mutation point
            for (int k = i; k < i + mutationPoint; k++) {
                first.add(firstSelected.getGenes().get(k));
            }

            // at mutation point flip values(turn on/off) for all the appliances at that hour
            String[] res = new String[21];
            res[0] = firstSelected.getGenes().get(i + mutationPoint)[0];
            for (int k = 1; k < res.length - 1; k++) {
                res[k] = "0.0";
            }
            res[20] = firstSelected.getGenes().get(i + mutationPoint)[20];
            first.add(res);

            // add to list the remaining hours of the day
            for (int k = i + mutationPoint + 1; k < i + 24; k++) {
                first.add(firstSelected.getGenes().get(k));
            }

        }

        // else add the consumptions of the day
        else {
            for (int k = i; k < i + 24; k++)
                first.add(firstSelected.getGenes().get(k));
        }
```

```
    }

    //iterate day per day
    for (int i = 0; i < secondSelected.getGenes().size(); i += 24) {

        // with possibility 1% turn on/off appliances at the first selected offspring at mutation point
        if (rn.nextInt( bound: 100) == 99) {

            // select a random mutation point in the day for the first selected
            int mutationPoint = rn.nextInt( bound: 24);

            // add to list until the mutation point
            for (int k = i; k < i + mutationPoint; k++) {
                second.add(secondSelected.getGenes().get(k));
            }

            // at mutation point flip values(turn on/off) for all the appliances at that hour
            String[] res = new String[21];
            res[0] = secondSelected.getGenes().get(i + mutationPoint)[0];
            for (int k = 1; k < res.length - 1; k++) {
                res[k] = "0.0";
            }
            res[20] = secondSelected.getGenes().get(i + mutationPoint)[20];
            second.add(res);

            // add to list the remaining hours of the day
            for (int k = i + mutationPoint + 1; k < i + 24; k++) {
                second.add(secondSelected.getGenes().get(k));
            }

        }

        // else add the consumptions of the day
        else {
            for (int k = i; k < i + 24; k++)
                second.add(secondSelected.getGenes().get(k));
        }
    }
```

```
    // save results
    firstSelected = new Chromosome(first);
    secondSelected = new Chromosome(second);


}

// get the fittest offspring (offspring with the least fitness value)
public Chromosome getFittestOffspring() {
    if ((best.getFitness() <= firstSelected.getFitness()) && (best.getFitness() <= secondSelected.getFitness()))
        return best;
    else if ((firstSelected.getFitness() <= secondSelected.getFitness())
            && (firstSelected.getFitness() <= best.getFitness())) {
        return firstSelected;
    } else {
        return secondSelected;
    }
}

// replace the least fit individual from the fittest offspring
public void addFittestOffspring() {

    // calculate fitness of the chromosomes
    firstSelected.calcFitness();
    secondSelected.calcFitness();
    best.calcFitness();

    // get index of the least fit individual
    int leastFittestIndex = population.getLeastFittestIndex();

    // replace the least fit individual from the fittest offspring
    population.getChromosomes()[leastFittestIndex] = new Chromosome(getFittestOffspring());

    firstSelected = null;
    secondSelected = null;
    best = null;
}
```

```java
// function to reallocate consumptions from peak hour to non-peak hour
public void peakHoursReallocation(List<String[]> peakHours) {

    // init variables
    int firstPeakHour, secondPeakHour, thirdPeakHour;
    float firstPeakHourConsumption, secondPeakHourConsumption, thirdPeakHourConsumption;
    int firstNonPeakHour, secondNonPeakHour, thirdNonPeakHour;
    float firstNonPeakHourConsumption, secondNonPeakHourConsumption, thirdNonPeakHourConsumption;
    int firstPeakGenerationHour, secondPeakGenerationHour, thirdPeakGenerationHour;
    float firstPeakGeneration, secondPeakGeneration, thirdPeakGeneration;
    String[][] checkSelected, checkBest;    // arrays for check the daily fitness after reallocation
    List<String[]> reallocatedData = new ArrayList<>();

    // iterate day by day
    for (int i = 0; i < firstSelected.getGenes().size(); i += 24) {

        // give the correct value to variables
        firstPeakHour = secondPeakHour = thirdPeakHour = 0;
        firstPeakHourConsumption = secondPeakHourConsumption = thirdPeakHourConsumption = Float.MIN_VALUE;
        firstNonPeakHour = secondNonPeakHour = thirdNonPeakHour = 0;
        firstNonPeakHourConsumption = secondNonPeakHourConsumption = thirdNonPeakHourConsumption = Float.MAX_VALUE;
        firstPeakGenerationHour = secondPeakGenerationHour = thirdPeakGenerationHour = 0;
        firstPeakGeneration = secondPeakGeneration = thirdPeakGeneration = Float.MIN_VALUE;
        checkBest = new String[24][21];
        checkSelected = new String[24][21];


        // find three peak hours and the three hours with the least consumption
        for (int j = i; j < i + 24; j++) {

            // copy values to the two arrays for check if result is better after reallocation
            for (int k = 0; k < best.getGenes().get(j).length; k++) {
                checkSelected[j % 24][k] = best.getGenes().get(j)[k];
                checkBest[j % 24][k] = best.getGenes().get(j)[k];
            }

            // replace comma with empty char
            String val = (peakHours.get(j)[2]).replaceAll( regex: ",",  replacement: "");

            // find three peak hours
            if (Float.parseFloat(val) > firstPeakHourConsumption) {
                thirdPeakHour = secondPeakHour;
                thirdPeakHourConsumption = secondPeakHourConsumption;
                secondPeakHour = firstPeakHour;
                secondPeakHourConsumption = firstPeakHourConsumption;
                firstPeakHour = j % 24;     // mod 24 to find the specific hour of the day
                firstPeakHourConsumption = Float.parseFloat(val);
            } else if (Float.parseFloat(val) > secondPeakHourConsumption) {
                thirdPeakHour = secondPeakHour;
                thirdPeakHourConsumption = secondPeakHourConsumption;
                secondPeakHour = j % 24;    // mod 24 to find the specific hour of the day
                secondPeakHourConsumption = Float.parseFloat(val);
            } else if (Float.parseFloat(val) > thirdPeakHourConsumption) {
                thirdPeakHour = j % 24;     // mod 24 to find the specific hour of the day
                thirdPeakHourConsumption = Float.parseFloat(val);
            }

            // find three non-peak hours
            if (Float.parseFloat(val) < firstNonPeakHourConsumption) {
                thirdNonPeakHour = secondNonPeakHour;
                thirdNonPeakHourConsumption = secondNonPeakHourConsumption;
                secondNonPeakHour = firstNonPeakHour;
                secondNonPeakHourConsumption = firstNonPeakHourConsumption;
                firstNonPeakHour = j % 24;
                firstNonPeakHourConsumption = Float.parseFloat(val);
            } else if (Float.parseFloat(val) < secondNonPeakHourConsumption) {
                thirdNonPeakHour = secondNonPeakHour;
                thirdNonPeakHourConsumption = secondNonPeakHourConsumption;
                secondNonPeakHour = j % 24;
                secondNonPeakHourConsumption = Float.parseFloat(val);
            } else if (Float.parseFloat(val) < thirdNonPeakHourConsumption) {
                thirdNonPeakHour = j % 24;
                thirdNonPeakHourConsumption = Float.parseFloat(val);
            }

            // find peak generation hours in the day
            if (Float.parseFloat(best.getGenes().get(j)[20]) > firstPeakGeneration) {
                thirdPeakGenerationHour = secondPeakGenerationHour;
                thirdPeakGeneration = secondPeakGeneration;
```

```
                secondPeakGenerationHour = firstPeakGenerationHour;
                secondPeakGeneration = firstPeakGeneration;
                firstPeakGenerationHour = j % 24;
                firstPeakGeneration = Float.parseFloat(best.getGenes().get(j)[20]);
            } else if (Float.parseFloat(best.getGenes().get(j)[20]) > secondPeakGeneration) {
                thirdPeakGenerationHour = secondPeakGenerationHour;
                thirdPeakGeneration = secondPeakGeneration;
                secondPeakGenerationHour = j % 24;
                secondPeakGeneration = Float.parseFloat(best.getGenes().get(j)[20]);
            } else if (Float.parseFloat(best.getGenes().get(j)[20]) > thirdPeakGeneration) {
                thirdPeakGenerationHour = j % 24;
                thirdPeakGeneration = Float.parseFloat(best.getGenes().get(j)[20]);
            }
        }

        // reallocate/swap appliances consumption in array for check
        for (int j = 0; j < checkBest[j % 24].length - 2; j++) {
            if ((firstPeakGenerationHour != firstPeakHour) && (secondPeakGenerationHour != firstPeakHour)
                    && (thirdPeakGenerationHour != firstPeakHour)) {
                String temp1 = checkBest[firstPeakHour][j + 1];
                checkBest[firstPeakHour][j + 1] = checkBest[firstNonPeakHour][j + 1];
                checkBest[firstNonPeakHour][j + 1] = temp1;
            }
            if ((firstPeakGenerationHour != secondPeakHour) && (secondPeakGenerationHour != secondPeakHour)
                    && (thirdPeakGenerationHour != secondPeakHour)) {
                String temp2 = checkBest[secondPeakHour][j + 1];
                checkBest[secondPeakHour][j + 1] = checkBest[secondNonPeakHour][j + 1];
                checkBest[secondNonPeakHour][j + 1] = temp2;
            }
            if ((firstPeakGenerationHour != thirdPeakHour) && (secondPeakGenerationHour != thirdPeakHour)
                    && (thirdPeakGenerationHour != thirdPeakHour)) {
                String temp3 = checkBest[thirdPeakHour][j + 1];
                checkBest[thirdPeakHour][j + 1] = checkBest[thirdNonPeakHour][j + 1];
                checkBest[thirdNonPeakHour][j + 1] = temp3;
            }
        }

        // compare the two arrays and find the best
        float checkSelectedResult = calculateImportedExported(checkSelected);
```

```
        float checkBestResult = calculateImportedExported(checkBest);

        // if best have better results reallocate
        if (checkBestResult < checkSelectedResult) {
            // if best after reallocation save the best results
            reallocatedData.addAll(Arrays.asList(checkBest).subList(0, 24));
        }

        // else save the previous results
        else {
            reallocatedData.addAll(Arrays.asList(checkSelected).subList(0, 24));
        }

    }

    // save the results
    best = new Chromosome(reallocatedData);

}

// function to compare the two consumptions
public float calculateImportedExported(String[][] check) {

    float Imp = 0.0F;
    float Exp = 0.0F;

    for (String[] hour : check) {

        // calculate hour consumption
        float hourConsumption = 0.0F;
        for (int j = 0; j < hour.length - 2; j++) {
            hourConsumption += (Float.parseFloat(hour[j + 1]));
        }

        // calculate imported/exported
        if (hourConsumption > Float.parseFloat(hour[20]))
            Imp += Math.abs(Float.parseFloat(hour[20]) - hourConsumption);
        else
            Exp += Math.abs(Float.parseFloat(hour[20]) - hourConsumption);
```

```
        }

        // return imported/exported based on their weight
        return Imp;
    }

    public float[][] calcConsumptions(List<String[]> genes) {
        float[][] consumptions = new float[366][19];

        // iterate above all data day per day
        for (int i = 0; i < genes.size(); i += 24) {

            // iterate per device in a day
            for (int j = 1; j < genes.get(i).length - 1; j++) {

                // count how many hours is open
                float counter = 0;
                for (int k = 0; k < 24; k++) {
                    counter += Float.parseFloat(genes.get(i + k)[j]);
                }

                // save the total consumption of the device
                consumptions[i / 24][j - 1] = counter;
            }
        }
        return consumptions;
    }

}
```

```
package cs.ucy.ac.cy;

import java.io.IOException;
import java.util.*;

public class ChromosomeComfort implements Cloneable {

    // object parameters init
    private float fitness;
    private float exportedEnergy;
    private float importedEnergy;
    private float totalConsumption;
    private float totalProduction;
    private float comfortCost;
    private List<String[]> genes;

    // chromosome constructor with the consumptions
    public ChromosomeComfort(List<String[]> energyConsumptions, float[] maxConsumption,
                             float[] averageConsumption) throws IOException {

        // set genes using random allocation of appliances
        this.genes = Algorithms.random( iterations: 1, energyConsumptions, maxConsumption, averageConsumption);
        this.fitness = 0.0F;
        this.exportedEnergy = 0.0F;
        this.importedEnergy = 0.0F;
        this.totalConsumption = 0.0F;
        this.totalProduction = 0.0F;
        this.comfortCost = 0.0F;
    }

    public ChromosomeComfort(List<String[]> list) {
        this.genes = list;
        this.fitness = 0.0F;
        this.exportedEnergy = 0.0F;
        this.importedEnergy = 0.0F;
        this.totalConsumption = 0.0F;
        this.totalProduction = 0.0F;
        this.comfortCost = 0.0F;
    }

```

```java
    // chromosome constructor with chromosome
    public ChromosomeComfort(ChromosomeComfort c) {
        this.genes = c.getGenes();
        this.fitness = c.getFitness();
        this.exportedEnergy = c.getExportedEnergy();
        this.importedEnergy = c.getImportedEnergy();
        this.totalConsumption = c.getTotalConsumption();
        this.totalProduction = c.getTotalProduction();
        this.comfortCost = c.getComfortCost();
    }

    // calculate fitness
    public void calcFitness(List<String[]> rules) {

        float totalImported = 0.0F;
        float totalExported = 0.0F;
        float totalConsumption = 0.0F;
        float totalProduction = 0.0F;
        float comfortCost = 0.0F;

        for (int i = 0; i < this.genes.size(); i++) {

            // calculate hour consumption
            float hourConsumption = 0.0F;
            for (int j = 0; j < this.genes.get(i).length - 2; j++) {
                float temp = Float.parseFloat(this.genes.get(i)[j + 1]);
                hourConsumption += (temp);
                for (int k = 0; k < rules.size(); k++) {
                    int appliance = Integer.parseInt(rules.get(k)[1]);
                    if (rules.get(k)[0].equals(this.genes.get(i)[0]) && appliance == j) {
                        if (this.genes.get(i)[appliance].equals("0.0")) {
                            comfortCost += 1000;
                        }
                    }
                }
            }

            totalConsumption += hourConsumption;
            totalProduction += Float.parseFloat(this.genes.get(i)[20]);

            // if consumption is greater than production energy equals imported
            if (hourConsumption > Float.parseFloat(this.genes.get(i)[20]))
                totalImported += (hourConsumption - Float.parseFloat(this.genes.get(i)[20]));
            else
                totalExported += (Float.parseFloat(this.genes.get(i)[20]) - hourConsumption);
        }

        // set values
//        this.fitness = totalImported;
        this.fitness = (float) (totalImported * 0.75) + (float) (0.25 * comfortCost);
        this.importedEnergy = totalImported;
        this.exportedEnergy = totalExported;
        this.totalConsumption = totalConsumption;
        this.totalProduction = totalProduction;
        this.comfortCost = comfortCost;
    }

    public void fixConsumptions(float[][] dailyConsumptions, float[] maxConsumption) {

        // threshold to check if two floats are equal
        final float THRESHOLD = (float) .00001;

        List<String[]> output = new ArrayList<>();

        // iterate day per day
        for (int i = 0; i < this.genes.size(); i += 24) {

            // copy in array the current day
            String[][] day = new String[24][21];
            for (int j = i; j < i + 24; j++) {
                System.arraycopy(this.genes.get(j),  srcPos: 0, day[j % 24],  destPos: 0,  length: 21);
            }

            // find how many times devices are open in the day after gen. alg.
            float[] checkConsumptions = new float[19];
            for (int j = 1; j < this.genes.get(i).length - 1; j++) {
                float counter = 0;
                for (int k = 0; k < 24; k++) {
                    counter += Float.parseFloat(day[k][j]);
```

```
            }
            checkConsumptions[j - 1] = counter;
        }

        Integer[] idx = new Integer[24];
        for (int j = 0; j < 24; j++) {
            idx[j] = j;
        }
        float[] productions = new float[24];
        for (int j = 0; j < 24; j++) {
            productions[j] = Float.parseFloat(day[j][20]);
        }

        Arrays.sort(idx, (o1, o2) -> Float.compare(productions[o1], productions[o2]));

        // fix the consumption by iterate device per device
        for (int j = 0; j < checkConsumptions.length; j++) {

            float consumptionDiff;

            if (Math.abs(checkConsumptions[j] - dailyConsumptions[i / 24][j]) < THRESHOLD) continue;
                // if is greater than the original turn off devices
            else if (checkConsumptions[j] > dailyConsumptions[i / 24][j]) {
                consumptionDiff = checkConsumptions[j] - dailyConsumptions[i / 24][j];
                for (Integer integer : idx) {

                    if (day[integer][j + 1].equals("0.0")) continue;
                    else {
                        if (consumptionDiff <= 0)
                            break;

                        float currCons = Float.parseFloat(day[integer][j + 1]);
                        if (currCons <= consumptionDiff) {
                            this.genes.get(i + integer)[j + 1] = "0.0";
                            consumptionDiff -= currCons;
                        } else {
                            float tmp = currCons - consumptionDiff;
                            day[integer][j + 1] = String.valueOf(tmp);
                            break;
```

```
                        }
                    }
                }
            }

            // if is less than the original turn on devices
            else if (checkConsumptions[j] < dailyConsumptions[i / 24][j]) {
                consumptionDiff = dailyConsumptions[i / 24][j] - checkConsumptions[j];
                for (int l = idx.length - 1; l >= 0; l--) {

                    if (consumptionDiff <= 0)
                        break;

                    float currCons = Float.parseFloat(day[idx[l]][j + 1]);
                    if ((currCons + consumptionDiff) >= maxConsumption[j]) {
                        day[idx[l]][j + 1] = String.valueOf(maxConsumption[j]);
                        consumptionDiff -= (maxConsumption[j] - currCons);
                    } else {
                        float tmp = currCons + consumptionDiff;
                        day[idx[l]][j + 1] = String.valueOf(tmp);
                        break;
                    }
                }
            }
        }

        Collections.addAll(output, day);
    }

    this.genes = new ArrayList<>(output);

}

@Override
protected Object clone() throws CloneNotSupportedException {
    ChromosomeComfort chromosome = (ChromosomeComfort) super.clone();
    chromosome.genes = new ArrayList<>();
    chromosome.genes.addAll(this.genes);
    chromosome.fitness = this.fitness;
```

```
        chromosome.importedEnergy = this.importedEnergy;
        chromosome.exportedEnergy = this.exportedEnergy;
        chromosome.totalConsumption = this.totalConsumption;
        chromosome.totalProduction = this.totalProduction;
        chromosome.comfortCost = this.comfortCost;
        return chromosome;
    }

    // fitness getter
    public float getFitness() { return fitness; }

    // exported energy getter
    public float getExportedEnergy() { return exportedEnergy; }

    // imported energy getter
    public float getImportedEnergy() { return importedEnergy; }

    // genes getter
    public List<String[]> getGenes() { return genes; }

    // total consumption getter
    public float getTotalConsumption() { return totalConsumption; }

    // total production getter
    public float getTotalProduction() { return totalProduction; }

    // total comfort cost getter
    public float getComfortCost() { return comfortCost; }
}
```

```
package cs.ucy.ac.cy;

import java.io.IOException;
import java.util.List;

public class PopulationComfort {

    // object parameters init
    private final int popSize;
    private ChromosomeComfort[] chromosomes;
    private float fittestScore = 0.0F;
    private float fittestExported = 0.0F;
    private float fittestImported = 0.0F;
    private float fittestConsumption = 0.0F;
    private float fittestProduction = 0.0F;
    private float fittestComfortCost = 0.0F;
    private int fittestIndex = -1;

    // population object constructor
    public PopulationComfort(int popSize, List<String[]> energyConsumptions,
                             float[] maxConsumption, float[] averageConsumption) throws IOException {
        super();
        this.popSize = popSize;
        this.chromosomes = new ChromosomeComfort[popSize];

        // create a first population pool
        for (int i = 0; i < popSize; i++) {
            chromosomes[i] = new ChromosomeComfort(energyConsumptions, maxConsumption, averageConsumption);
        }
    }

    // get the fittest individual (individual with the lowest fitness)
    public void getFittest() {
        float maxFit = Float.MAX_VALUE;
        int maxFitIndex = 0;
        for (int i = 0; i < chromosomes.length; i++) {
            if (chromosomes[i].getFitness() < maxFit) {
                maxFit = chromosomes[i].getFitness();
                maxFitIndex = i;
            }
        }
```

```
        fittestScore = chromosomes[maxFitIndex].getFitness();
        fittestImported = chromosomes[maxFitIndex].getImportedEnergy();
        fittestExported = chromosomes[maxFitIndex].getExportedEnergy();
        fittestConsumption = chromosomes[maxFitIndex].getTotalConsumption();
        fittestProduction = chromosomes[maxFitIndex].getTotalProduction();
        fittestComfortCost = chromosomes[maxFitIndex].getComfortCost();
        fittestIndex = maxFitIndex;

    }

    // get index of the least fit individual
    public int getLeastFittestIndex() {
        float minFitVal = Float.MIN_VALUE;
        int minFitIndex = 0;
        for (int i = 0; i < chromosomes.length; i++) {
            if (chromosomes[i].getFitness() > minFitVal) {
                minFitVal = chromosomes[i].getFitness();
                minFitIndex = i;
            }
        }
        return minFitIndex;
    }

    // calculate fitness of each individual
    public void calculateFitness(List<String[]> rules) {

        for (ChromosomeComfort chromosome : chromosomes) {
            chromosome.calcFitness(rules);
        }
        getFittest();
    }

    // population size getter
    public int getPopSize() { return popSize; }

    // individuals getter
    public ChromosomeComfort[] getChromosomes() { return chromosomes; }

    // fittest score getter
    public float getFittestScore() { return fittestScore; }

    public float getFittestExported() { return fittestExported; }

    public float getFittestImported() { return fittestImported; }

    public float getFittestConsumption() { return fittestConsumption; }

    public float getFittestProduction() { return fittestProduction; }

    public float getFittestComfortCost() { return fittestComfortCost; }

    public int getFittestIndex() { return fittestIndex; }
}
```

```java
package cs.ucy.ac.cy;

import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;
import java.util.Random;

public class GreenCapComfort {

    // object parameters init
    private final PopulationComfort population;
    private ChromosomeComfort firstSelected;
    private ChromosomeComfort secondSelected;
    private ChromosomeComfort best;
    private int generationCount;
    Random rn = new Random();

    // GreenCap object constructor
    public GreenCapComfort(List<String[]> energyConsumptions, int popSize,
                           float[] maxConsumption, float[] averageConsumption) throws IOException {
        this.population = new PopulationComfort(popSize, energyConsumptions, maxConsumption, averageConsumption);
        this.generationCount = 0;
    }

    // function to implement allocation of appliances using genetic algorithm
    public static List<String[]> GeneticAlgorithmComfort(List<String[]> energyConsumptions, List<String[]> peakHours,
                                                         float[] maxConsumption, float[] averageConsumption,
                                                         List<String[]> rules) throws IOException {

        // parameters init
        final int popSize = 30;
        final int maxGeneration = 10;

        // init GreenCap object - Population
        GreenCapComfort greenCapComfort = new GreenCapComfort(energyConsumptions, popSize,
                maxConsumption, averageConsumption);

        // calculate how total consumption of a device in a day for all days of the year
        float[][] dailyConsumption = greenCapComfort.calcConsumptions(energyConsumptions);

        System.out.println("\nGreenCapComfort:");
        System.out.println("Population of " + greenCapComfort.population.getPopSize() + " chromosome(s).");

        //Calculate fitness of each individual
        greenCapComfort.population.calculateFitness(rules);

        // print current generation and fitness
        System.out.println("Generation: " + greenCapComfort.generationCount + " Fittest Score: " +
                greenCapComfort.population.getFittestScore() + " Exported Energy: " +
                greenCapComfort.population.getFittestExported() + " kWh. Imported Energy: " +
                greenCapComfort.population.getFittestImported() + " kWh. Total Production: " +
                greenCapComfort.population.getFittestProduction() + " kWh. Total Consumption: " +
                greenCapComfort.population.getFittestConsumption() + " kWh. Comfort Cost: " +
                greenCapComfort.population.getFittestComfortCost());

        // while current generation is less than max number of generations
        while (greenCapComfort.generationCount < maxGeneration) {

            int iteration = 0;

            // increase the current generation
            ++greenCapComfort.generationCount;

            // while iteration is less than the number of individuals in population
            while (iteration < popSize) {

                // do selection (random select two individuals)
                greenCapComfort.selection();

                // do crossover function
                greenCapComfort.crossover();

                // do mutation function
                greenCapComfort.mutation();

                // fix the consumptions to match the first consumption
                greenCapComfort.firstSelected.fixConsumptions(dailyConsumption, maxConsumption);
                greenCapComfort.secondSelected.fixConsumptions(dailyConsumption, maxConsumption);
```

```
            // update fitness values of offspring and clone best offspring
            greenCapComfort.firstSelected.calcFitness(rules);
            greenCapComfort.secondSelected.calcFitness(rules);
            try {
                if (greenCapComfort.firstSelected.getFitness() < greenCapComfort.secondSelected.getFitness())
                    greenCapComfort.best = (ChromosomeComfort) greenCapComfort.firstSelected.clone();
                else
                    greenCapComfort.best = (ChromosomeComfort) greenCapComfort.secondSelected.clone();
            } catch (CloneNotSupportedException e) {
                e.printStackTrace();
            }

            // reallocate appliances from peak to non-peak hour
            greenCapComfort.peakHoursReallocation(peakHours);

            // add the fittest offspring to population
            greenCapComfort.addFittestOffspring(rules);

            // increase iteration
            ++iteration;
        }

        // find the fittest chromosome
        greenCapComfort.population.getFittest();

        // print current generation and fitness
        System.out.println("Generation: " + greenCapComfort.generationCount + " Fittest Score: " +
                greenCapComfort.population.getFittestScore() + " Exported Energy: " +
                greenCapComfort.population.getFittestExported() + " kWh. Imported Energy: " +
                greenCapComfort.population.getFittestImported() + " kWh. Total Production: " +
                greenCapComfort.population.getFittestProduction() + " kWh. Total Consumption: " +
                greenCapComfort.population.getFittestConsumption() + " kWh. Comfort Cost: " +
                greenCapComfort.population.getFittestComfortCost());

    }

    System.out.println("Fitness: " + greenCapComfort.population.getFittestScore());
```

```
    // fix consumption for every chromosome
    for (ChromosomeComfort c : greenCapComfort.population.getChromosomes())
        c.fixConsumptions(dailyConsumption, maxConsumption);

    greenCapComfort.population.calculateFitness(rules);
    greenCapComfort.population.getFittest();

    return greenCapComfort.population.getChromosomes()[greenCapComfort.population.getFittestIndex()].getGenes();

}

// selection function
public void selection() {

    try {
        // random select the first individual
        int firstRandom = rn.nextInt(population.getChromosomes().length);
        firstSelected = (ChromosomeComfort) population.getChromosomes()[firstRandom].clone();

        // random select the second individual (make sure second != first)
        int secondRandom = rn.nextInt(population.getChromosomes().length);
        while (secondRandom == firstRandom) {
            secondRandom = rn.nextInt(population.getChromosomes().length);
        }
        secondSelected = (ChromosomeComfort) population.getChromosomes()[secondRandom].clone();
    } catch (CloneNotSupportedException e) {
        e.printStackTrace();
    }

}

// crossover function
public void crossover() {

    //int crossOverPoint = rn.nextInt(24);

    // lists to save the results for crossover
    List<String[]> first = new ArrayList<>();
    List<String[]> second = new ArrayList<>();
```

```
        // iterate day per day
        for (int i = 0; i < firstSelected.getGenes().size(); i += 24) {

            // with possibility 90% swap values between the two offsprings at crossover point
            if (rn.nextInt( bound: 100) <= 90) {

                // select a random crossover point in the day
                int crossOverPoint = rn.nextInt( bound: 24);

                // swap consumption until the crossover point between the two offsprings
                for (int k = i; k < i + crossOverPoint; k++) {
                    first.add(secondSelected.getGenes().get(k));
                    second.add(firstSelected.getGenes().get(k));
                }

                // fill the list with the remaining hours of the day
                for (int k = i + crossOverPoint; k < i + 24; k++) {
                    first.add(firstSelected.getGenes().get(k));
                    second.add(secondSelected.getGenes().get(k));
                }
            }

            // else just fill with the current consumptions
            else {
                for (int k = i; k < i + 24; k++) {
                    first.add(firstSelected.getGenes().get(k));
                    second.add(secondSelected.getGenes().get(k));
                }
            }
        }

        // save results
        firstSelected = new ChromosomeComfort(first);
        secondSelected = new ChromosomeComfort(second);

    }
```

```
    // mutation function
    public void mutation() {

        // select a random mutation point in the day for the first selected
        //int mutationPoint = rn.nextInt(24);

        // lists to save the results for mutation
        List<String[]> first = new ArrayList<>();
        List<String[]> second = new ArrayList<>();

        //iterate day per day
        for (int i = 0; i < firstSelected.getGenes().size(); i += 24) {

            // with possibility 1% turn on/off appliances at the first selected offspring at mutation point
            if (rn.nextInt( bound: 100) == 99) {

                // select a random mutation point in the day for the first selected
                int mutationPoint = rn.nextInt( bound: 24);

                // add to list until the mutation point
                for (int k = i; k < i + mutationPoint; k++) {
                    first.add(firstSelected.getGenes().get(k));
                }

                // at mutation point flip values(turn on/off) for all the appliances at that hour
                String[] res = new String[21];
                res[0] = firstSelected.getGenes().get(i + mutationPoint)[0];
                for (int k = 1; k < res.length - 1; k++) {
                    res[k] = "0.0";
                }
                res[20] = firstSelected.getGenes().get(i + mutationPoint)[20];
                first.add(res);

                // add to list the remaining hours of the day
                for (int k = i + mutationPoint + 1; k < i + 24; k++) {
                    first.add(firstSelected.getGenes().get(k));
                }

            }
```

```
            // else add the consumptions of the day
            else {
                for (int k = i; k < i + 24; k++)
                    first.add(firstSelected.getGenes().get(k));
            }
        }

        //iterate day per day
        for (int i = 0; i < secondSelected.getGenes().size(); i += 24) {

            // with possibility 1% turn on/off appliances at the first selected offspring at mutation point
            if (rn.nextInt( bound: 100) == 99) {

                // select a random mutation point in the day for the first selected
                int mutationPoint = rn.nextInt( bound: 24);

                // add to list until the mutation point
                for (int k = i; k < i + mutationPoint; k++) {
                    second.add(secondSelected.getGenes().get(k));
                }

                // at mutation point flip values(turn on/off) for all the appliances at that hour
                String[] res = new String[21];
                res[0] = secondSelected.getGenes().get(i + mutationPoint)[0];
                for (int k = 1; k < res.length - 1; k++) {
                    res[k] = "0.0";
                }
                res[20] = secondSelected.getGenes().get(i + mutationPoint)[20];
                second.add(res);

                // add to list the remaining hours of the day
                for (int k = i + mutationPoint + 1; k < i + 24; k++) {
                    second.add(secondSelected.getGenes().get(k));
                }

            }

            // else add the consumptions of the day
            else {
                for (int k = i; k < i + 24; k++)
                    second.add(secondSelected.getGenes().get(k));
            }
        }

        // save results
        firstSelected = new ChromosomeComfort(first);
        secondSelected = new ChromosomeComfort(second);


    }

    // get the fittest offspring (offspring with the least fitness value)
    public ChromosomeComfort getFittestOffspring() {
        if ((best.getFitness() <= firstSelected.getFitness()) && (best.getFitness() <= secondSelected.getFitness())) {
            return best;
        } else if ((firstSelected.getFitness() <= secondSelected.getFitness())
                && (firstSelected.getFitness() <= best.getFitness())) {
            return firstSelected;
        } else {
            return secondSelected;
        }
    }

    // replace the least fit individual from the fittest offspring
    public void addFittestOffspring(List<String[]> rules) {

        // calculate fitness of the chromosomes
        firstSelected.calcFitness(rules);
        secondSelected.calcFitness(rules);
        best.calcFitness(rules);

        // get index of the least fit individual
        int leastFittestIndex = population.getLeastFittestIndex();

        // replace the least fit individual from the fittest offspring
        population.getChromosomes()[leastFittestIndex] = new ChromosomeComfort(getFittestOffspring());

        firstSelected = null;
        secondSelected = null;
```

```
        best = null;
    }

    // function to reallocate consumptions from peak hour to non-peak hour
    public void peakHoursReallocation(List<String[]> peakHours) {

        // init variables
        int firstPeakHour, secondPeakHour, thirdPeakHour;
        float firstPeakHourConsumption, secondPeakHourConsumption, thirdPeakHourConsumption;
        int firstNonPeakHour, secondNonPeakHour, thirdNonPeakHour;
        float firstNonPeakHourConsumption, secondNonPeakHourConsumption, thirdNonPeakHourConsumption;
        int firstPeakGenerationHour, secondPeakGenerationHour, thirdPeakGenerationHour;
        float firstPeakGeneration, secondPeakGeneration, thirdPeakGeneration;
        String[][] checkSelected, checkBest;    // arrays for check the daily fitness after reallocation
        List<String[]> reallocatedData = new ArrayList<>();

        // iterate day by day
        for (int i = 0; i < firstSelected.getGenes().size(); i += 24) {

            // give the correct value to variables
            firstPeakHour = secondPeakHour = thirdPeakHour = 0;
            firstPeakHourConsumption = secondPeakHourConsumption = thirdPeakHourConsumption = Float.MIN_VALUE;
            firstNonPeakHour = secondNonPeakHour = thirdNonPeakHour = 0;
            firstNonPeakHourConsumption = secondNonPeakHourConsumption = thirdNonPeakHourConsumption = Float.MAX_VALUE;
            firstPeakGenerationHour = secondPeakGenerationHour = thirdPeakGenerationHour = 0;
            firstPeakGeneration = secondPeakGeneration = thirdPeakGeneration = Float.MIN_VALUE;
            checkBest = new String[24][21];
            checkSelected = new String[24][21];


            // find three peak hours and the three hours with the least consumption
            for (int j = i; j < i + 24; j++) {

                // copy values to the two arrays for check if result is better after reallocation
                for (int k = 0; k < best.getGenes().get(j).length; k++) {
                    checkSelected[j % 24][k] = best.getGenes().get(j)[k];
                    checkBest[j % 24][k] = best.getGenes().get(j)[k];
                }

                // replace comma with empty char
                String val = (peakHours.get(j)[2]).replaceAll( regex: ",",  replacement: "");

                // find three peak hours
                if (Float.parseFloat(val) > firstPeakHourConsumption) {
                    thirdPeakHour = secondPeakHour;
                    thirdPeakHourConsumption = secondPeakHourConsumption;
                    secondPeakHour = firstPeakHour;
                    secondPeakHourConsumption = firstPeakHourConsumption;
                    firstPeakHour = j % 24;     // mod 24 to find the specific hour of the day
                    firstPeakHourConsumption = Float.parseFloat(val);
                } else if (Float.parseFloat(val) > secondPeakHourConsumption) {
                    thirdPeakHour = secondPeakHour;
                    thirdPeakHourConsumption = secondPeakHourConsumption;
                    secondPeakHour = j % 24;    // mod 24 to find the specific hour of the day
                    secondPeakHourConsumption = Float.parseFloat(val);
                } else if (Float.parseFloat(val) > thirdPeakHourConsumption) {
                    thirdPeakHour = j % 24;     // mod 24 to find the specific hour of the day
                    thirdPeakHourConsumption = Float.parseFloat(val);
                }

                // find three non-peak hours
                if (Float.parseFloat(val) < firstNonPeakHourConsumption) {
                    thirdNonPeakHour = secondNonPeakHour;
                    thirdNonPeakHourConsumption = secondNonPeakHourConsumption;
                    secondNonPeakHour = firstNonPeakHour;
                    secondNonPeakHourConsumption = firstNonPeakHourConsumption;
                    firstNonPeakHour = j % 24;
                    firstNonPeakHourConsumption = Float.parseFloat(val);
                } else if (Float.parseFloat(val) < secondNonPeakHourConsumption) {
                    thirdNonPeakHour = secondNonPeakHour;
                    thirdNonPeakHourConsumption = secondNonPeakHourConsumption;
                    secondNonPeakHour = j % 24;
                    secondNonPeakHourConsumption = Float.parseFloat(val);
                } else if (Float.parseFloat(val) < thirdNonPeakHourConsumption) {
                    thirdNonPeakHour = j % 24;
                    thirdNonPeakHourConsumption = Float.parseFloat(val);
                }

```

```
            // find peak generation hours in the day
            if (Float.parseFloat(best.getGenes().get(j)[20]) > firstPeakGeneration) {
                thirdPeakGenerationHour = secondPeakGenerationHour;
                thirdPeakGeneration = secondPeakGeneration;
                secondPeakGenerationHour = firstPeakGenerationHour;
                secondPeakGeneration = firstPeakGeneration;
                firstPeakGenerationHour = j % 24;
                firstPeakGeneration = Float.parseFloat(best.getGenes().get(j)[20]);
            } else if (Float.parseFloat(best.getGenes().get(j)[20]) > secondPeakGeneration) {
                thirdPeakGenerationHour = secondPeakGenerationHour;
                thirdPeakGeneration = secondPeakGeneration;
                secondPeakGenerationHour = j % 24;
                secondPeakGeneration = Float.parseFloat(best.getGenes().get(j)[20]);
            } else if (Float.parseFloat(best.getGenes().get(j)[20]) > thirdPeakGeneration) {
                thirdPeakGenerationHour = j % 24;
                thirdPeakGeneration = Float.parseFloat(best.getGenes().get(j)[20]);
            }
        }

        // reallocate/swap appliances consumption in array for check
        for (int j = 0; j < checkBest[j % 24].length - 2; j++) {
            if ((firstPeakGenerationHour != firstPeakHour) && (secondPeakGenerationHour != firstPeakHour)
                    && (thirdPeakGenerationHour != firstPeakHour)) {
                String temp1 = checkBest[firstPeakHour][j + 1];
                checkBest[firstPeakHour][j + 1] = checkBest[firstNonPeakHour][j + 1];
                checkBest[firstNonPeakHour][j + 1] = temp1;
            }
            if ((firstPeakGenerationHour != secondPeakHour) && (secondPeakGenerationHour != secondPeakHour)
                    && (thirdPeakGenerationHour != secondPeakHour)) {
                String temp2 = checkBest[secondPeakHour][j + 1];
                checkBest[secondPeakHour][j + 1] = checkBest[secondNonPeakHour][j + 1];
                checkBest[secondNonPeakHour][j + 1] = temp2;
            }
            if ((firstPeakGenerationHour != thirdPeakHour) && (secondPeakGenerationHour != thirdPeakHour)
                    && (thirdPeakGenerationHour != thirdPeakHour)) {
                String temp3 = checkBest[thirdPeakHour][j + 1];
                checkBest[thirdPeakHour][j + 1] = checkBest[thirdNonPeakHour][j + 1];
                checkBest[thirdNonPeakHour][j + 1] = temp3;
            }
```

```
        }

        // compare the two arrays and find the best
        float checkSelectedResult = calculateImportedExported(checkSelected);
        float checkBestResult = calculateImportedExported(checkBest);

        // if best have better results reallocate
        if (checkBestResult < checkSelectedResult) {
            // if best after reallocation save the best results
            reallocatedData.addAll(Arrays.asList(checkBest).subList(0, 24));
        }

        // else save the previous results
        else {
            reallocatedData.addAll(Arrays.asList(checkSelected).subList(0, 24));
        }

    }

    // save the results
    best = new ChromosomeComfort(reallocatedData);

}

// function to compare the two consumptions
public float calculateImportedExported(String[][] check) {

    float Imp = 0.0F;
    float Exp = 0.0F;

    for (String[] hour : check) {

        // calculate hour consumption
        float hourConsumption = 0.0F;
        for (int j = 0; j < hour.length - 2; j++) {
            hourConsumption += (Float.parseFloat(hour[j + 1]));
        }
```

```
            // calculate imported/exported
            if (hourConsumption > Float.parseFloat(hour[20]))
                Imp += Math.abs(Float.parseFloat(hour[20]) - hourConsumption);
            else
                Exp += Math.abs(Float.parseFloat(hour[20]) - hourConsumption);

        }

        // return imported/exported based on their weight
        return Imp;
    }

    public float[][] calcConsumptions(List<String[]> genes) {
        float[][] consumptions = new float[366][19];

        // iterate above all data day per day
        for (int i = 0; i < genes.size(); i += 24) {

            // iterate per device in a day
            for (int j = 1; j < genes.get(i).length - 1; j++) {

                // count how many hours is open
                float counter = 0;
                for (int k = 0; k < 24; k++) {
                    counter += Float.parseFloat(genes.get(i + k)[j]);
                }

                // save the total consumption of the device
                consumptions[i / 24][j - 1] = counter;
            }
        }
        return consumptions;
    }

}
```