

Ατομική Διπλωματική Εργασία

**Υλοποίηση και Αποτίμηση του Πλαισίου Εκτέλεσης Κανόνων
Προτιμήσεων σε Έξυπνα Κτήρια**

Αντώνης Βασιλείου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ



ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

**Υλοποίηση και Αποτίμηση του Πλαισίου Εκτέλεσης Κανόνων Προτιμήσεων σε
Έξυπνα Κτήρια**

Αντώνης Βασιλείου

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 2021

Ευχαριστίες

Αρχικά, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου Δρ. Δημήτρη Ζεϊναλιπούρ του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου, για την καθοδήγηση του καθ' όλη τη διάρκεια εκπόνησης αυτής της πτυχιακής εργασίας. Λόγω των αλλόκοτων συνθηκών της πανδημίας στην εκπαίδευση μας, παρουσιάστηκαν δυσκολίες. Παρόλα αυτά μέσω των συχνών συναντήσεων που είχαμε καταφέραμε να πετύχουμε τους στόχους που τέθηκαν. Ακόμη θα ήθελα να τον ευχαριστήσω για τις γνώσεις που απέκτησα στη διάρκεια της συνεργασίας μας.

Θα ήθελα να δώσω ένα μεγάλο ευχαριστώ στην οικογένεια μου για όλη την οικονομική και ψυχολογική υποστήριξη που μου έχει δώσει καθ' όλη τη πορεία μου στο Πανεπιστήμιο Κύπρου. Η οικογένεια μου με στήριξε σε όλες τις δυσκολίες είχα με το μέγιστο δυνατό τρόπο και αυτό με βοήθησε να μην τα παρατήσω ποτέ.

Επίσης θα ήθελα να ευχαριστήσω όλους τους φίλους μου εντός και εκτός του Πανεπιστημίου Κύπρου που στάθηκαν δίπλα μου και με υποστήριξαν.

Τέλος, θα ήθελα να ευχαριστώ τον Σωτήρη Κωνσταντίνου, διδακτορικό φοιτητή του Εργαστηρίου Συστημάτων Διαχείρισης Δεδομένων στο Τμήμα Πληροφορικής του Πανεπιστημίου Κύπρου, για τη συμβολή του στην υποστήριξη και καθοδήγηση της παρούσας διπλωματικής εργασίας.

Η επίδειξη του συστήματος IMCF που περιγράφεται σε αυτήν τη διπλωματική εργασία έχει παρουσιαστεί στο πιο κάτω συνέδριο:

- "IMCF: The IoT Meta-Control Firewall for Smart Buildings", Soteris Constantinou, Antonis Vasileiou, Andreas Konstantinidis, Panos K. Chrysanthis, Demetrios Zeinalipour-Yazti, 24th International Conference on Extending Database Technology (EDBT'21), OpenProceedings.org, pp. 658--661, March 23 - March 26, 2021, Nicosia, Cyprus, 2021

Περίληψη

Η υπερθέρμανση του πλανήτη είναι ένα γεγονός το οποίο έχει αναγνωριστεί από όλες τις χώρες παγκόσμια. Τα τελευταία χρόνια γίνονται τεράστιες προσπάθειες από όλα τα κράτη για μείωση ως και εξαφάνιση αυτού του φαινομένου. Η τεράστια ανάπτυξη του Διαδικτύου των πραγμάτων την τελευταία εικοσαετία, έχει αποφέρει ένα καινούργιο τρόπο διαχείρισης των έξυπνων συσκευών σε ένα έξυπνο κτήριο. Αυτές οι συσκευές μπορούν να ενεργοποιηθούν και να απενεργοποιηθούν ανάλογα με τις προτιμήσεις του χρήστη, μέσω ροών αυτοματισμού εργασίας. Με στόχο την μείωση της περιττής κατανάλωσης ηλεκτρικής ενέργειας καθώς και της μείωσης της εκπομπής διοξειδίου του άνθρακα, έχει υλοποιηθεί και αποτιμηθεί το πλαίσιο ικανοποίησης κανόνων προτιμήσεων σε έξυπνα κτήρια.

Στην παρούσα διπλωματική εργασία έχουν υλοποιηθεί οι αλγόριθμοι Energy Planner και Green Planner, οι οποίοι χρησιμοποιούν μέτα-κανόνες από ένα πίνακα μέτα-κανόνων που ορίζονται από χρήστες του συστήματος. Στους κανόνες μπορούν να συμπεριληφθούν προτιμήσεις για τη θερμοκρασία και το επίπεδο φωτός ανά διάστημα ώρας και ένα προτιμώμενο όριο κατανάλωσης ενέργειας. Βάσει αυτών των κανόνων οι αλγόριθμοι προσπαθούν να διατηρήσουν αυτές τις προτιμήσεις και να διατηρήσουν το προτιμώμενο όριο ενέργειας που έχει οριστεί δημιουργώντας ένα πλάνο ενεργοποίησης και απενεργοποίησης των κανόνων. Επίσης ο αλγόριθμος Green Planner προσπαθεί να κρατήσει τις εκπομπές διοξειδίου του άνθρακα στο ελάχιστο. Αφού οριστεί το πλάνο οι αλγόριθμοι συνεχίζουν με την ενεργοποίηση και απενεργοποίηση των διαφόρων συσκευών μέσω ροών αυτοματισμού για την διατήρηση των προτιμήσεων του χρήστη.

Οι αλγόριθμοι βάσει των αποτιμήσεων κατάφεραν να μειώσουν την κατανάλωση ενέργειας κατά 10% από το προτιμώμενο όριο που έχουν θέσει οι χρήστες, ενώ ταυτόχρονα παραμένουν άνετοι κατά 93% στην χειρότερη περίπτωση και 98% στην καλύτερη. Συνεπώς οι αλγόριθμοι που υλοποιήθηκαν κατάφεραν να επιτύχουν το στόχο της μείωσης της κατανάλωσης ενέργειας και της εκπομπής ρίπων θερμοκηπίου.

Περιεχόμενα

Κεφάλαιο 1 Εισαγωγή	1
1.1 Παρακίνηση.....	1
1.2 Πρόβλημα	2
1.3 Συνεισφορά.....	2
1.4 Περίγραμμα εργασίας.....	3
Κεφάλαιο 2 Σχετική Εργασία	5
2.1 Έξυπνα συστήματα διαχείρισης ενέργειας.....	6
2.1.1 Διαχειριζόμενη οικιακή ενέργεια με φωτοβολταϊκά	6
2.1.2 Έξυπνοι θερμοστάτες.....	7
2.1.3 Έξυπνες πρίζες με παρακολούθηση ενέργειας	8
2.1.4 Στοιβα πρωτοκόλλου EEBUS.....	8
2.2 Ροές αυτοματισμού εργασίας (RAW).....	9
2.2.1 IFTTT σε πραγματικό χρόνο	10
2.2.2 Μέθοδοι αναζήτησης με ενημέρωση RAW.....	10
2.3 Έξυπνη ενεργοποίηση.....	11
2.4 Διαφορές έξυπνων συστημάτων με το σύστημα που αναπτύχθηκε.....	12
2.5 Διαχείριση ζήτησης ενέργειας.....	12
2.6 Αλγόριθμοι.....	13
2.6.1 Local Search	13
2.6.2 Αλγόριθμοι Hill Climbing	14
2.6.3 Simulated Annealing.....	16
Κεφάλαιο 3 Αρχιτεκτονική	18
3.1 Τοπικός Ελεγκτής (LC).....	18
3.2 Το σύστημα IMCF	19
3.3 Δεδομένα εισόδου.....	20
3.4 Ο αλγόριθμος IMCF	21

3.4.1 Σχέδιο απόσβεσης (AP)	22
3.4.2 Αρχιτεκτονική αλγόριθμου ενεργειακού σχεδίου (EP)	24
3.4.3 Αρχιτεκτονική αλγόριθμου Green Planner (GP)	26
3.5 Οι βασικές προσεγγίσεις.....	28
3.5.1 Βασική προσέγγιση χωρίς κανόνες (NR)	29
3.5.2 Βασική προσέγγιση με χρήση όλων των μέτα-κανόνων (MR)	29
Κεφάλαιο 4 Υλοποίηση Αλγορίθμων	30
4.1 Εισαγωγή dataset.....	30
4.2 Ενημέρωση των μέτα-κανόνων από το σύστημα IMCF.....	31
4.3 Εισαγωγή των μέτα-κανόνων στο σύστημα	32
4.4 Υλοποίηση αλγόριθμου noRule / noIFTTT	33
4.5 Υλοποίηση του αλγόριθμου Energy Planner.....	34
4.6 Υλοποίηση του αλγόριθμου Green Planner	37
Κεφάλαιο 5 Διεπαφές Διαπροσωπείας	40
5.1 Δημιουργία λογαριασμού	40
5.2 Εισαγωγή του χρήστη στο σύστημα.....	41
5.3 Σελίδα Διαχείρισης Μέτα-Κανόνων.....	42
5.4 Εισαγωγή κανόνα Μέτα-Κανόνα.....	42
5.4 Επεξεργασία Μέτα-Κανόνα	43
5.6 Συσκευές συνδεδεμένες στο σύστημα	44
5.7 Αποτελέσματα εκτέλεσης	46
Κεφάλαιο 6 Πειραματική Μεθοδολογία & Αποτίμηση.....	48
6.1 Πλατφόρμα	48
6.2 Σύνολα δεδομένων	49
6.3 Μετρικές	51
6.4 Αλγόριθμοι.....	51
6.4.1 Αλγόριθμοι NR και MR.....	51

6.4.2 Αλγόριθμος If-This-Then-That (IFTTT)	52
6.4.3 Αλγόριθμος Energy Planner	52
6.4.4 Αλγόριθμος Green Planner	52
6.5 Αξιολόγηση απόδοσης	53
6.6 Συμπεράσματα από την πειραματική εφαρμογή.....	55
Κεφάλαιο 7 Συμπεράσματα	57
7.1 Συμπεράσματα	57
7.2 Μελλοντική δουλειά	58
Αναφορές	59
Παράρτημα Α	1

Κατάλογος Σχημάτων

Σχήμα 2.1 Sunny Home Manager Schema [4]	7
Σχήμα 2.2 Έξυπνος Θερμοστάτης τύπου Nest [6].....	7
Σχήμα 2.3 Έξυπνη πρίζα [7]	8
Σχήμα 2.4 EEBUS stack [9]	9
Σχήμα 2.5 Παράδειγμα ροής αυτοματισμού εργασίας [11].....	10
Σχήμα 2.6 Έξυπνος ενεργοποιητής περιστροφικής κίνησης [14].....	12
Σχήμα 2.7 Επιφάνεια με 2 τοπικά μέγιστα [17].....	14
Σχήμα 3.1 Ο αλγόριθμος IMCF με χρήση της EP [11]	24
Σχήμα 3.2 Παράδειγμα εκτέλεσης του IMCF με EP	26
Σχήμα 3.3 Ο αλγόριθμος IMCF με χρήση της GP [21].....	26
Σχήμα 3.4 Παράδειγμα εκτέλεσης του IMCF με GP.....	28
Σχήμα 5.1 Δημιουργία Λογαριασμού	41
Σχήμα 5.2 Εισαγωγή στο σύστημα	41
Σχήμα 5.3 Κανόνες Meta-Rules	42
Σχήμα 5.4 Δημιουργία κανόνα Meta-Rule	43
Σχήμα 5.5 Επεξεργασία κανόνα Meta-Rule	44
Σχήμα 5.6 Έξυπνες συσκευές συνδεδεμένες στο σύστημα	45
Σχήμα 5.7 Αποτελέσματα Εκτέλεσης 1	46
Σχήμα 5.8 Αποτελέσματα Εκτέλεσης 2	47
Σχήμα 5.9 Αποτελέσματα Green Planner	47
Σχήμα 6.1 Αξιολόγηση απόδοσης Energy Planner [4]	54
Σχήμα 6.2 Αξιολόγηση απόδοσης Green Planner [21].....	55

Κατάλογος Πινάκων

Πίνακας 2.1 Τύποι Διαχείρισης Ενέργειας [15]	13
Πίνακας 2.2 Παραλλαγές Hill Climbing [17]	15
Πίνακας 3.1 Παράδειγμα ECP [11]	21
Πίνακας 6.1 Κανόνες IFTTT [11]	52
Πίνακας 6.2 Εξοικονόμηση Χρημάτων με χρήση των αλγόριθμων.....	54

Κεφάλαιο 1

Εισαγωγή

1.1 Παρακίνηση.....	1
1.2 Πρόβλημα	2
1.3 Συνεισφορά.....	2
1.4 Περίγραμμα εργασίας.....	3

1.1 Παρακίνηση

Η παγκόσμια υπερθέρμανση του πλανήτη είναι ένα πλέον γνωστό φαινόμενο κατά το οποίο προβλέπονται διάφορες παγκόσμιες καταστροφές εάν δεν υπάρξει κάποια λύση. Ευτυχώς εδώ και κάποια χρόνια υπήρξε κινητοποίηση από όλα τα έθνη για λύση αυτού του σοβαρού προβλήματος.

Το 2016 με το σύμφωνο “Paris Agreement” [1] το οποίο είναι μια νομικά δεσμευτική διεθνής συνθήκη για την αλλαγή του κλίματος, τέθηκε σε κίνηση. Στόχος του είναι να μειώσει την αύξηση της σε λιγότερο από 2 βαθμούς Κελσίου. Για να επιτύχουν αυτόν τον μακροπρόθεσμο στόχο θερμοκρασίας, οι χώρες στοχεύουν στην επίτευξη της παγκόσμιας κορυφής των εκπομπών αερίων θερμοκηπίου το συντομότερο δυνατό για να επιτύχουν έναν κλιματικά ουδέτερο κόσμο έως τα μέσα του αιώνα. Αυτή η συμφωνία αποτελεί ορόσημο της πολυμερούς διαδικασίας αλλαγής του κλίματος, διότι για πρώτη φορά δεσμεύονται με μια συμφωνία όλα τα έθνη, για την καταπολέμηση της κλιματικής αλλαγής. Για την εκτέλεση αυτής της συμφωνίας απαιτούνται οικονομικές και κοινωνικές αλλαγές.

Η ευρωπαϊκή ένωση με το “Green deal” [2] στοχεύει μέχρι το 2050 να είναι η πρώτη ήπειρος που δεν θα έχει ρίπους αερίων του θερμοκηπίου. Για να το πετύχει αυτό δόθηκε ένα σχέδιο δράσης κατά το οποίο θα ενισχυθεί η αποτελεσματική χρήση των πόρων για μετάβαση σε μια καθαρή και κυκλική οικονομία, καθώς και η αποκατάσταση της βιοποικιλότητας με μείωση της ρύπανσης.

Ο διαγωνισμός με όνομα “Saves” [3], είναι ένας διαγωνισμός που πραγματοποιήθηκε κατά τα ακαδημαϊκά έτη 2014/15 και 2015/16, στον οποίον οι φοιτητές/τριες παρακινήθηκαν να εξοικονομήσουν ενέργεια δημιουργώντας έναν αγώνα μεταξύ κοιτώνων, όπου ανταγωνίζονταν στην εξοικονόμηση της περισσότερης ενέργειας. Στόχος αυτού του έργου ήταν η εγκατάσταση συνηθειών εξοικονόμησης ενέργειας στους μαθητές σε μια βασική στιγμή αλλαγής στη ζωή τους, ώστε να μπορούν να συνεχίσουν τις ενέργειες εξοικονόμησης ενέργειας καθ’ όλη τη διάρκεια της ιδιωτικής τους ζωής.

Με έμπνευση τις παραπάνω κινητοποιήσεις και με σκοπό την εξοικονόμηση ενέργειας και την μείωση των ρίπων του διοξειδίου του άνθρακα, συντάχτηκε η ακόλουθη ατομική διπλωματική εργασία.

1.2 Πρόβλημα

Τα σημερινά συστήματα διαχείρισης ενέργειας στον χώρο του διαδικτύου των πραγμάτων είναι πολύ ανεπτυγμένα. Για παράδειγμα υπάρχουν δυνατότητες προγραμματισμού ενεργοποίησης και απενεργοποίησης των διαφόρων έξυπνων συσκευών σε διάφορες ώρες της ημέρας ή ανάλογα με τον καιρό. Δυστυχώς όμως δεν υπάρχει δυνατότητα σε κανένα σύστημα διαχείρισης συσκευών σε ένα έξυπνο περιβάλλον, κατά την οποία μπορούν να τεθούν και να τηρηθούν μακροχρόνιοι στόχοι εξοικονόμησης ενέργειας.

1.3 Συνεισφορά

Το Πλαίσιο Εκτέλεσης Κανόνων Προτιμήσεων για έξυπνα κτήρια είναι ένα καινοτόμο σύστημα το οποίο στοχεύει να καλύψει το κενό του χειροκίνητου συντονισμού ροών αυτοματισμού κανόνων RAW για την επίτευξη των στόχων κατανάλωσης ενέργειας. Αρχικά ο χρήστης ή μια ομάδα χρηστών καθορίζει ένα διάνυσμα ροών αυτοματισμού

κανόνων σε ένα πίνακα μέτα-κανόνων που ονομάζεται Meta-Rule-Table (MRT) και ενός προφίλ κατανάλωσης ενέργειας που ονομάζεται ECP. Στόχος του είναι να προσδιοριστεί μεταξύ όλων των κανόνων MRT εκείνων που πρέπει να απορριφθούν, ώστε ο χρήστης να παραμείνει εντός του επιθυμητού ενεργειακού προϋπολογισμού σύμφωνα με το ιστορικό του ECP. Συγκεκριμένα το σύστημα χρησιμοποιεί τον αλγόριθμο Energy Planner (EP) climbing και τον αλγόριθμο Green Planner (GP), ο οποίος είναι εμπνευσμένος από αλγόριθμους τεχνητής νοημοσύνης όπως hill climbing και simulated annealing αντίστοιχα. Ο αλγόριθμος περνά πάνω στον μεγάλο χώρο αναζήτησης N-συνδυασμών όπου το N είναι το σύνολο των κανόνων, αποδίδοντας γρήγορα τους κανόνες που πρέπει να απορριφθούν για να επιτευχθεί ο στόχος του χρήστη. Το πλαίσιο προσαρμόζει τις ροές αυτοματισμού με τέτοιο τρόπο ώστε να μην έρχονται σε αντίθεση με τους μακροπρόθεσμους στόχους των χρηστών, απορρίπτοντας ορισμένους κανόνες βάσει προτεραιότητας προτιμήσεων. Σημαντικό να αναφερθεί ότι οι ροές αυτοματισμού κανόνων διακρίνονται σε δύο κατηγορίες άνεσης και αναγκαιότητας. Οι κανόνες ευκολίας στοχεύουν στην προώθηση της φυσικής άνεσης ενός ατόμου π.χ. θερμοκρασία δωματίου, φωτισμός περιβάλλοντος ή ό, τι θεωρείται προσωρινή άνεση, ενώ οι κανόνες αναγκαιότητας είναι εκείνοι οι κανόνες που πρέπει πάντα να εκτελούνται ανεξάρτητα από το εάν επιτυγχάνεται ο μακροπρόθεσμος στόχος.

1.4 Περίγραμμα εργασίας

Η παρούσα διπλωματική εργασία αποτελείται από επτά κεφάλαια. Αυτά είναι η «Εισαγωγή», η «Σχετική Δουλειά», η «Αρχιτεκτονική», το «Υλοποίηση Αλγορίθμων», το «Διεπαφές Διαπροσωπείας», η «Πειραματική Μεθοδολογία & Αποτίμηση» και τα «Συμπεράσματα».

Σε αυτό το κεφάλαιο αναφέρθηκαν οι λόγοι που ώθησαν την υλοποίηση του πλαισίου εκτέλεσης κανόνων προτιμήσεων σε έξυπνα κτήρια. Επίσης, αναφέρθηκε η συνεισφορά που προσφέρει το πλαίσιο αυτό.

Στο δεύτερο κεφάλαιο ακολουθεί περιγραφή σχετικών δουλειών που έχουν γίνει στο πεδίο ενασχόλησης της εργασίας αυτής, των ροών αυτοματισμού ελέγχου και των αλγορίθμων στους οποίους βασίζεται η υλοποίηση.

Στο τρίτο κεφάλαιο δίνεται μια λεπτομερής περιγραφή της αρχιτεκτονικής του συστήματος στο οποίο έχει αναπτυχθεί το πλαίσιο μας, τα δεδομένα εισόδου του συστήματος και οι αλγόριθμοι οι οποίοι αναπτύχθηκαν.

Στο τέταρτο κεφάλαιο δίνεται μια λεπτομερής περιγραφή της υλοποίησης των αλγορίθμων του πλαισίου, η οποία περιλαμβάνει τους αλγόριθμους Energy Planner, Green Planner, no-Rule, Meta-Rule και IFTTT.

Στο πέμπτο κεφάλαιο περιγράφεται η γραφική διαπροσωπεία του πλαισίου, στο οποίο μπορεί ένας χρήστης να δει τα αποτελέσματα του αλγόριθμου Energy Planner ή Green Planner αναλόγως, καθώς και να προσθέσει ή να αφαιρέσει κανόνες.

Στο έκτο κεφάλαιο περιγράφεται η μεθοδολογία που χρησιμοποιήθηκε για την εκτέλεση των διαφόρων πειραμάτων. Συγκεκριμένα περιγράφεται το testbed στο οποίο έγιναν τα πειράματα, οι μετρικές που χρησιμοποιήθηκαν και οι αλγόριθμοι που αξιολογήθηκαν. Επίσης περιγράφεται η αξιολόγηση της απόδοσης και τα συμπεράσματα που βγήκαν από την πειραματική αποτίμηση.

Τέλος, στο έβδομο κεφάλαιο αναφέρονται τα γενικά συμπεράσματα που προκύπτουν από την παρούσα διπλωματική εργασία, όπως και παράθεση κάποιων πιθανών μελλοντικών εργασιών για περεταίρω ανάπτυξη του πλαισίου.

Κεφάλαιο 2

Σχετική Εργασία

2.1 Έξυπνα συστήματα διαχείρισης ενέργειας	6
2.1.1 Διαχειριζόμενη οικιακή ενέργεια με φωτοβολταϊκά	6
2.1.2 Έξυπνοι θερμοστάτες.....	7
2.1.3 Έξυπνες πρίζες με παρακολούθηση ενέργειας	8
2.1.4 Στοιβα πρωτοκόλλου EEBUS.....	8
2.2 Ροές αυτοματισμού εργασίας (RAW).....	9
2.2.1 IFTTT σε πραγματικό χρόνο	10
2.2.2 Μέθοδοι αναζήτησης με ενημέρωση RAW.....	10
2.3 Έξυπνη ενεργοποίηση.....	11
2.4 Διαφορές έξυπνων συστημάτων με το σύστημα που αναπτύχθηκε.....	12
2.5 Διαχείριση ζήτησης ενέργειας.....	12
2.6 Αλγόριθμοι.....	13
2.6.1 Local Search	13
2.6.2 Αλγόριθμοι Hill Climbing	14
2.6.3 Simulated Annealing.....	16

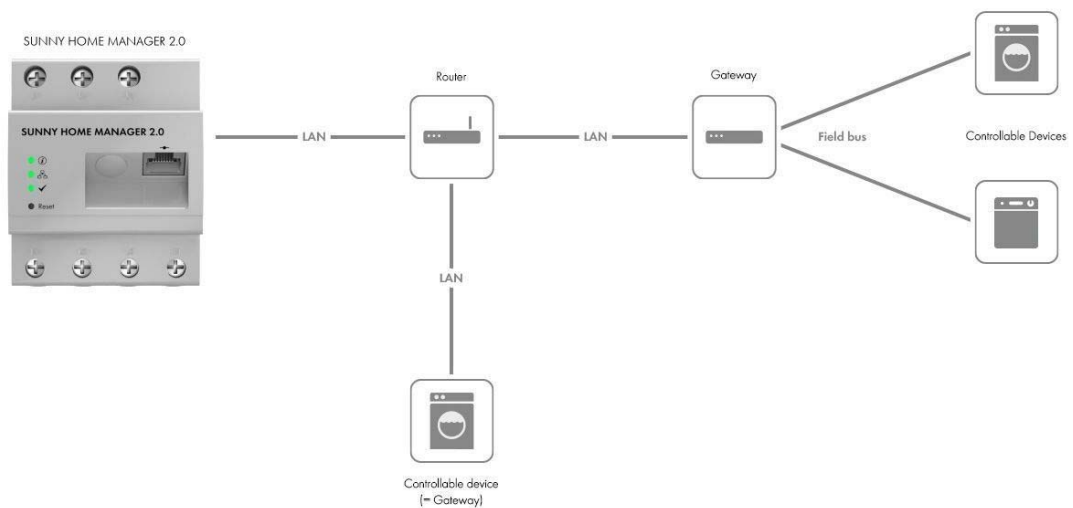
Σε αυτό το κεφάλαιο θα δούμε εν συντομία διάφορες τεχνολογίες οι οποίες χρησιμοποιούνται σήμερα για την διαχείριση έξυπνων συσκευών καθώς και τις αυτών των συστημάτων με το σύστημα που αναπτύχθηκε εντός αυτής της διπλωματικής εργασίας.

2.1 Έξυπνα συστήματα διαχείρισης ενέργειας

Η υιοθέτηση του διαδικτύου των πραγμάτων τα τελευταία χρόνια έφερε ένα αναζωογονημένο ενδιαφέρον για τη διαχείριση δεδομένων και τις λύσεις μηχανικής δεδομένων, τις αρχιτεκτονικές και τις εφαρμογές με έμφαση στην απορρόφηση δεδομένων, αναλυτικές αρχιτεκτονικές για ροή δεδομένων καθώς και σχετική συγκριτική αξιολόγηση. Κάποιες λύσεις αναφέρονται παρακάτω.

2.1.1 Διαχειριζόμενη οικιακή ενέργεια με φωτοβολταϊκά

Οι ελεγκτές Sunny Home Manager [4] από την SMA είναι συνδεδεμένοι με όλες τις συσκευές του σπιτιού. Μπορούν να διαχειριστούν την κατανάλωση ηλεκτρικού ρεύματος από τις συσκευές και να χρησιμοποιήσουν την παραγόμενη ηλεκτρική ενέργεια από τα φωτοβολταϊκά. Συγκεκριμένα παρακολουθούν τις ροές ισχύος, ιδιαίτερα την παραγωγή ισχύος AC από τους μετατροπείς και την κατανάλωση ισχύος AC από τα νοικοκυριά (καταγράφεται από έναν μετρητή ενέργειας). Στη συνέχεια, διαχειρίζεται ανάλογα τους φόρτους εργασίας κατανάλωσης ισχύος (π.χ. πότε πρέπει να λειτουργεί πλυντήριο ρούχων ή ο έξυπνος φορτιστής αυτοκινήτου, έτσι ώστε η αυτοκατανάλωση ηλιακής ενέργειας να βελτιστοποιείται). Οι ελεγκτές SAM χρησιμοποιούν το ανοικτό πρωτόκολλο απλής διαχείρισης ενέργειας (SEMP) [5] ή το EEBUS που αναφέρεται στη συνέχεια.



Σχήμα 2.1 Sunny Home Manager Schema [4]

2.1.2 Έξυπνοι θερμοστάτες

Ένα καλό παράδειγμα έξυπνου θερμοστάτη, είναι ο θερμοστάτης εκμάθησης Google Nest [6], είναι ένας προγραμματιζόμενος και αυτοδίδακτος θερμοστάτης με δυνατότητα Wi-Fi που βελτιστοποιεί την ψύξη και τη θέρμανση για εξοικονόμηση ενέργειας. Επίσης μπορεί να ειδοποιήσει με ηλεκτρονικό μήνυμα αν εντοπίσει πως χρειάζεται περισσότερο χρόνο από τον κανονικό για την θέρμανση ή ψύξη του σπιτιού.



Σχήμα 2.2 Έξυπνος Θερμοστάτης τύπου Nest [6]

2.1.3 Έξυπνες πρίζες με παρακολούθηση ενέργειας

Οι πρίζες τύπου “Kasa Smart WiFi Plug Slim with Energy Monitoring” [7] της εταιρίας tp-link μπορούν να ενεργοποιηθούν από εφαρμογή στο κινητό, μπορούν να προστεθούν προγράμματα ενεργοποίησης και απενεργοποίησης και με την εφαρμογή στο κινητό σου δείχνει την κατανάλωση ενέργειας που έχουν ξοδέψει οι συσκευές που είναι συνδεδεμένες μέσω τέτοιων πριζών.

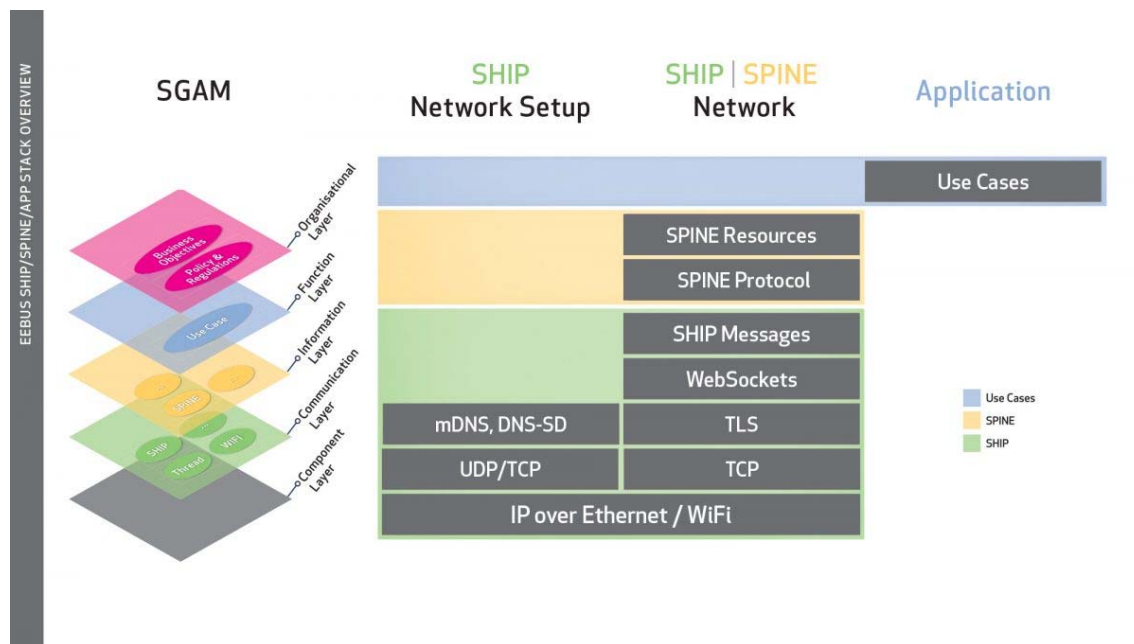


Σχήμα 2.3 Έξυπνη πρίζα [7]

2.1.4 Στοιίβα πρωτοκόλλου EEBUS

Η στοιίβα πρωτοκόλλου EEBUS [8] χρησιμοποιείται στο Διαδίκτυο των πραγμάτων για την επικοινωνία μεταξύ καταναλωτών ηλεκτρικής ενέργειας, παραγωγών, αποθηκών και διαχειριστικών οντοτήτων. Η σουίτα πρωτοκόλλου βασίζεται και αναπτύσσεται πάνω από το υφιστάμενο πρωτόκολλο διαδικτύου, είναι εξαιρετικά γενικό, μπορεί να εφαρμοστεί μεταξύ σε τομείς και είναι ανοιχτό και δωρεάν στο κοινό. Ο κύριος τομέας εφαρμογής του πρωτοκόλλου είναι η διαχείριση όμως ζήτησης ενέργειας, η ανταλλαγή δεδομένων και ο έλεγχος των συσκευών, όμως μπορεί να εφαρμοστεί για οικιακό αυτοματισμό.

Η αρχιτεκτονική του EEBUS βασίζεται στο αρχιτεκτονικό μοντέλο SGAM. Στο παρακάτω σχήμα φαίνεται η στοιίβα EEBUS.

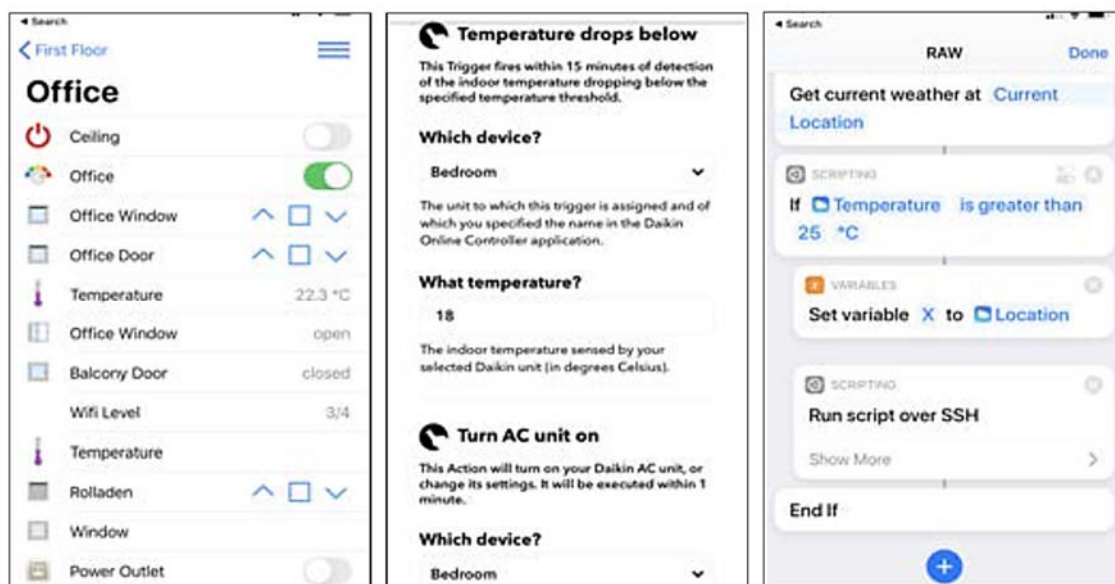


Σχήμα 2.4 EEBUS stack [9]

2.2 Ροές αυτοματισμού εργασίας (RAW)

Οι ροές εργασίας [10] επιτρέπουν των αυτοματισμό συγκεκριμένων καθημερινών εργασιών, ώστε να μην γίνονται χειροκίνητα. Μια ροή εργασίας (κανόνας αυτοματισμού) ορίζεται από κανόνες πυροδότησης (πότε θα ξεκινήσει ο κανόνας), συνθήκες (τα κριτήρια αντιστοίχισης για μια εντολή) και ενέργειες (τι εφαρμόζεται σε μια εντολή που ταιριάζει με τους καθορισμένους κανόνες ετικέτας).

Ένα παράδειγμα ροών εργασίας είναι οι "Κανόνες αποστολής". Οι κανόνες αποστολής επιτρέπουν την αντιστοιχία μεταφορέα, τη μέθοδο αποστολής και τον τύπο πακέτου με βάση την αντίστοιχη χώρα και πολιτεία.



Σχήμα 2.5 Παράδειγμα ροής αυτοματισμού εργασίας [11]

Οι ροές αυτοματισμού εργασίας (RAW) [11], εκτείνονται από απλές δηλώσεις έως διαδικαστικές ροές εργασίας με στόχο τη σύλληψη ενός έξυπνου αγωγού ενεργοποίησης σε εργαλεία που ελέγχουν τις διάφορες έξυπνες συσκευές.

2.2.1 IFTTT σε πραγματικό χρόνο

Το RT-IFTTT [12], είναι μια γλώσσα διαδικτύου των πραγμάτων όπου κατά την εφαρμογή της, χρησιμοποιεί εύκαμπτα διαστήματα ανίχνευσης αισθητήρων συνθηκών ενεργοποίησης σε πραγματικό χρόνο. Η γλώσσα RTIFTTT επεκτείνει την υπάρχουσα σύνταξη IFTTT και επιτρέπει στους χρήστες να καθορίσουν περιορισμούς σε πραγματικό χρόνο για τα τις έξυπνες συσκευές τους.

2.2.2 Μέθοδοι αναζήτησης με ενημέρωση RAW

Οι μέθοδοι RAW [11] χαρακτηρίζονται γενικά από ένα βοηθητικό πρόγραμμα στη σάρωση του χώρου λύσεων για την επίτευξη του στόχου. Αυτοί οι αλγόριθμοι

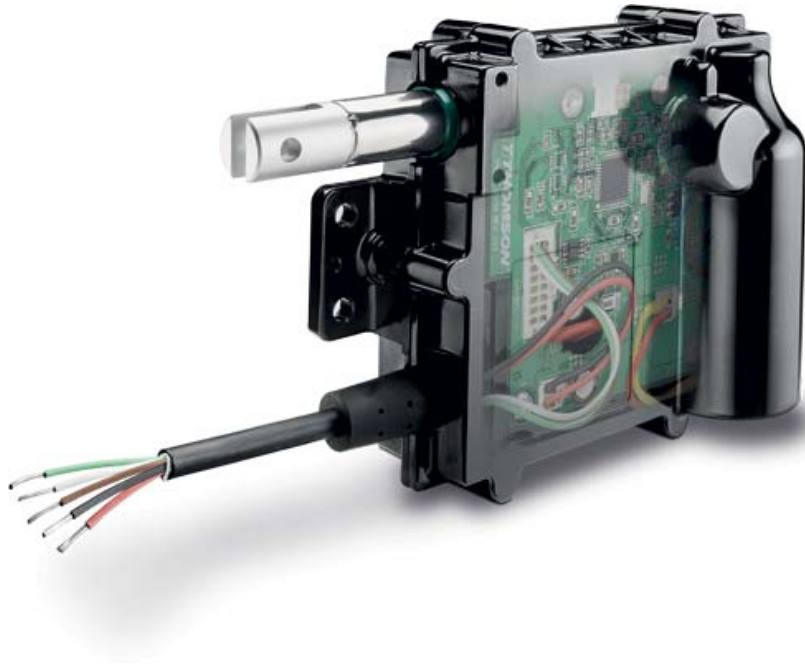
χρησιμοποιούν κάποιο είδος συνάρτησης αξιολόγησης που αξιολογεί λανθασμένα κάποια απόσταση της τρέχουσας λύσης από έναν στόχο.

2.3 Έξυπνη ενεργοποίηση

Οι ενεργοποιητές [13] είναι μηχανικές συσκευές που επιτρέπουν την εφαρμογή μιας συγκεκριμένης κίνησης ή δύναμης έμμεσα σε ένα προϊόν ή μια διαδικασία, και όχι χειροκίνητα. Ο έλεγχος των ενεργοποιητών μπορεί χειροκίνητα ή με τη βοήθεια λογισμικού ή μέσω από διεπαφές σε ένα υπολογιστή. Οι ενεργοποιητές που μπορούν να προγραμματιστούν είναι γνωστοί ως «έξυπνοι» ενεργοποιητές. Δύο συνήθεις τύποι ενεργοποιητών περιλαμβάνουν τη γραμμική ή την ευθεία κίνηση και την περιστροφική ή την κυκλική κίνηση.

Τα οφέλη [14] χρήσης έξυπνων ενεργοποιητών περιλαμβάνουν:

- Αυξημένη αποδοτικότητα και παραγωγικότητα.
- Βελτιωμένες διαγνωστικές δυνατότητες και δυνατότητα ελέγχου.
- Λιγότερα εξαρτήματα και λιγότερη καλωδίωση.
- Ελαχιστοποιημένη πολυπλοκότητα και ευκολότερη εγκατάσταση.
- Μειωμένο κόστος υλικού και λογισμικού.
- Μειωμένος χρόνος και βάρος ανάπτυξης μηχανών.
- Βελτιωμένη λειτουργικότητα και απόδοση του μηχανήματος.



Σχήμα 2.6 Έξυπνος ενεργοποιητής περιστροφικής κίνησης [14]

2.4 Διαφορές έξυπνων συστημάτων με το σύστημα που αναπτύχθηκε

Τα παραπάνω έξυπνα συστήματα διαχείρισης ενέργειας χρησιμοποιούν πρωτόκολλα τα οποία είναι προσανατολισμένα για την διαχείριση φορτίου μέσα σε έξυπνα κτήρια, χωρίς όμως να θέτουν μακροχρόνιους στόχους κατανάλωσης ενέργειας. Αντίθετα η υλοποίηση των αλγορίθμων Energy Planner και Green Planner που αναφέρονται στο κεφάλαιο 4, χρησιμοποιούν κανόνες που θέτονται από τους χρήστες για επίτευξη μακροπρόθεσμου στόχου κατανάλωσης ενέργειας.

2.5 Διαχείριση ζήτησης ενέργειας

Η διαχείριση ζήτησης ενέργειας [15], γνωστή και ως διαχείριση από πλευράς ζήτησης (DSM) ή απόκριση από πλευράς ζήτησης (DSR), είναι η τροποποίηση της ζήτησης ενέργειας για τους καταναλωτές μέσω διαφόρων μεθόδων, όπως οικονομικά κίνητρα και αλλαγή συμπεριφοράς μέσω της εκπαίδευσης.

Συνηθής στόχος της διαχείρισης ζήτησης είναι να ενθαρρύνει τον καταναλωτή να χρησιμοποιεί λιγότερη ενέργεια κατά τις ώρες αιχμής ή να μεταφέρει τον χρόνο χρήσης

ενέργειας σε ώρες εκτός αιχμής, όπως τη νύχτα και τα σαββατοκύριακα. Η μέγιστη διαχείριση της ζήτησης δεν μειώνει απαραίτητα τη συνολική κατανάλωση ενέργειας, αλλά αναμένεται να μειώσει την ανάγκη για επενδύσεις σε δίκτυα ή / και σταθμούς παραγωγής ενέργειας για την κάλυψη των αιχμών.

Με το σύστημα που αναπτύχθηκε επί τους σκοπούς της ατομικής διπλωματικής εργασίας ο στόχος της διαχείρισης ζήτησης, γίνεται πιο εφικτός διότι μειώνεται γενικότερα η κατανάλωση ενεργείας.

Types of DSM
Energy efficiency
Demand response
Dynamic demand
Distributed Energy Resources

Πίνακας 2.1 Τύποι Διαχείρισης Ενέργειας [15]

2.6 Αλγόριθμοι

Σε αυτή την υποενότητα αναφέρονται οι αλγόριθμοι στους οποίους βασίζονται οι αλγόριθμοι Energy Planner και Green Planner που υλοποιήθηκαν στα πλαίσια αυτής της διπλωματικής εργασίας.

2.6.1 Local Search

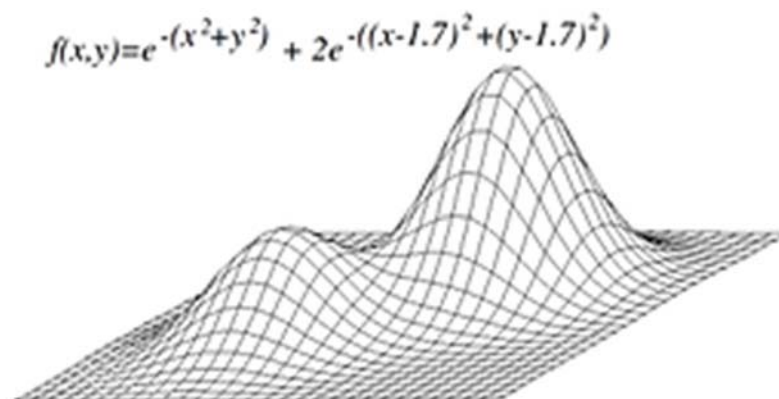
Η τοπική αναζήτηση [16] είναι μια ευρετική μέθοδος για την επίλυση υπολογιστικών σκληρών προβλημάτων βελτιστοποίησης. Η τοπική αναζήτηση μπορεί να χρησιμοποιηθεί σε προβλήματα που μπορούν να διατυπωθούν ως εξεύρεση λύσης μεγιστοποιώντας ένα κριτήριο μεταξύ ορισμένων υποψηφίων λύσεων. Οι τοπικοί αλγόριθμοι αναζήτησης μετακινούνται από λύση σε λύση στο χώρο των υποψηφίων λύσεων (ο χώρος αναζήτησης) εφαρμόζοντας τοπικές αλλαγές, έως ότου βρεθεί μια λύση που θεωρείται βέλτιστη ή έχει παρέλθει ένα χρονικό όριο.

Τα περισσότερα προβλήματα μπορούν να διατυπωθούν από άποψη χώρου αναζήτησης και στόχου με διάφορους τρόπους. Για παράδειγμα, για το πρόβλημα του πωλητή που ταξιδεύει, μια λύση μπορεί να είναι ένας κύκλος και το κριτήριο της μεγιστοποίησης είναι ένας συνδυασμός του αριθμού των κόμβων και της διάρκειας του κύκλου.

2.6.2 Αλγόριθμοι Hill Climbing

Hill climbing [17] είναι μια μαθηματική μέθοδος βελτίωσης η οποία ανήκει στην κατηγορία των τοπικών αναζητήσεων που αναφέρθηκαν πιο πάνω. Είναι ένας επαναληπτικός αλγόριθμος ο οποίος αρχίζει με μια αρχική λύση και σε προσπαθεί να βρει μια καλύτερη λύση κάνοντας μια αυξητική αλλαγή στην αρχική λύση. Ο αλγόριθμος συνεχίζει να προσθέτει αλλαγές μέχρι να μην υπάρχουν άλλες αυξητικές αλλαγές.

Γενικά οι τεχνικές τύπου Hill climbing είναι πολύ καλές σε κυρτά προβλήματα τα οποία έχουν ένα μέγιστο, δηλαδή μόνο μια καλύτερη λύση. Σε προβλήματα που υπάρχουν τοπικά μέγιστα ο αλγόριθμος δεν θα βρει απαραίτητα ως καλύτερη λύση το ολικό μέγιστο αλλά είναι πολύ πιθανόν να βρει ένα αυτά ως καλύτερη λύση. Για την λύση αυτού του προβλήματος χρησιμοποιούνται κάποιες τεχνικές όπως επανεκκινήσεις και simulated annealing.



Σχήμα 2.7 Επιφάνεια με 2 τοπικά μέγιστα [17]

Ένα παράδειγμα χρήσης είναι στο πρόβλημα του πλανόδιου πωλητή [18]. Ο αλγόριθμος ξεκινώντας από μια μη βέλτιστη λύση και κάνει μικρές βελτιώσεις σε αυτή, όπως η αλλαγή της σειράς με την οποία επισκέπτονται δύο πόλεις και τελικά μια πολύ μικρότερη διαδρομή είναι πιθανό να επιτευχθεί.

Αλγόριθμος Simple Hill Climbing [19]:

Βήμα 1: Αξιολογήστε την αρχική κατάσταση, εάν είναι κατάσταση στόχου, τότε επιστρέψτε την επιτυχία και σταματήστε.

Βήμα 2: Βρόχος μέχρι να βρεθεί λύση ή δεν υπάρχει νέος χειριστής για να υποβάλει αίτηση.

Βήμα 3: Επιλέξτε και εφαρμόστε έναν τελεστή στην τρέχουσα κατάσταση.

Βήμα 4: Έλεγχος νέας κατάστασης:

Εάν είναι κατάσταση στόχου, επιστρέψτε την επιτυχία και σταματήστε.

Διαφορετικά, εάν είναι καλύτερη από την τρέχουσα κατάσταση, τότε εκχωρήστε νέα κατάσταση ως τρέχουσα κατάσταση.

Διαφορετικά, αν όχι καλύτερα από την τρέχουσα κατάσταση, επιστρέψτε στο βήμα 2.

Βήμα 5: Έξοδος.

Στον πίνακα παρακάτω αναφέρονται διάφορες παραλλαγές του αλγορίθμου hill climbing.

Παραλλαγές της Μεθόδου Hill Climbing
Simple hill climbing
Stochastic hill climbing
Steepest ascent hill climbing
Random-restart hill climbing

Πίνακας 2.2 Παραλλαγές Hill Climbing [17]

2.6.3 Simulated Annealing

Η προσομοιωμένη ανόπτηση [20] (SA) είναι μια πιθανολογική τεχνική για την προσέγγιση του ολικού βέλτιστου όγκου μιας δεδομένης λειτουργίας. Συγκεκριμένα, είναι μια μεταευριστική προσέγγιση της ολικής βελτιστοποίησης σε ένα μεγάλο χώρο αναζήτησης για ένα πρόβλημα βελτιστοποίησης. Χρησιμοποιείται συχνά όταν ο χώρος αναζήτησης είναι διακριτός (π.χ., το πρόβλημα του πλανόδιου πωλητή).

Το όνομα του αλγορίθμου προέρχεται από την ανόπτηση στη μεταλλουργία, μια τεχνική που περιλαμβάνει θέρμανση και ελεγχόμενη ψύξη ενός υλικού για την αύξηση του μεγέθους των κρυστάλλων του και τη μείωση των ελαττωμάτων τους. Η προσομοιωμένη ανόπτηση μπορεί να χρησιμοποιηθεί για πολύ σκληρά προβλήματα υπολογιστικής βελτιστοποίησης όπου οι ακριβείς αλγόριθμοι αποτυγχάνουν, παρόλο που συνήθως επιτυγχάνει μια κατά προσέγγιση λύση στο ελάχιστο σε παγκόσμιο επίπεδο, θα μπορούσε να είναι αρκετή για πολλά πρακτικά προβλήματα.

Τα προβλήματα που επιλύονται από την SA διατυπώνονται επί του παρόντος από μια αντικειμενική συνάρτηση πολλών μεταβλητών, που υπόκεινται σε διάφορους περιορισμούς. Στην πράξη, ο περιορισμός μπορεί να τιμωρηθεί ως μέρος της αντικειμενικής λειτουργίας.

Σε κάθε βήμα, η SA ευρετική εξετάζει κάποια γειτονική κατάσταση s^* της τρέχουσας κατάστασης και αποφασίζει πιθανώς μεταξύ της μετακίνησης του συστήματος στην κατάσταση s^* ή της παραμονής στην κατάσταση. Αυτές οι πιθανότητες οδηγούν τελικά το σύστημα να μετακινηθεί σε καταστάσεις χαμηλότερης ενέργειας. Συνήθως αυτό το βήμα επαναλαμβάνεται έως ότου το σύστημα φτάσει σε μια κατάσταση που είναι αρκετά καλή για την εφαρμογή ή έως ότου εξαντληθεί ένας δεδομένος προϋπολογισμός υπολογισμού.

Η βελτιστοποίηση μιας λύσης περιλαμβάνει την αξιολόγηση των γειτόνων μιας κατάστασης του προβλήματος, οι οποίες είναι νέες καταστάσεις που παράγονται μέσω της συντηρητικής αλλαγής μιας δεδομένης κατάστασης. Για παράδειγμα, στο πρόβλημα του πλανόδιου πωλητή κάθε κράτος ορίζεται συνήθως ως παραλλαγή των πόλεων προς επίσκεψη, και οι γείτονες οποιασδήποτε πολιτείας είναι το σύνολο των παραλλαγών που

παράγονται με την ανταλλαγή δύο από αυτές τις πόλεις. Ο καλά καθορισμένος τρόπος με τον οποίο οι καταστάσεις αλλάζουν για να παράγουν γειτονικά κράτη ονομάζεται «κίνηση» και διαφορετικές κινήσεις δίνουν διαφορετικά σύνολα γειτονικών κρατών. Αυτές οι κινήσεις συνήθως οδηγούν σε ελάχιστες αλλαγές της τελευταίας κατάστασης, σε μια προσπάθεια βελτίωσης σταδιακά της λύσης μέσω της επαναληπτικής βελτίωσης των τμημάτων της (όπως οι συνδέσεις πόλεων στο πρόβλημα του ταξιδιώτη πωλητή).

Προκειμένου να εφαρμοστεί η προσομοιωμένη μέθοδος ανόπτησης σε ένα συγκεκριμένο πρόβλημα, πρέπει να προσδιοριστούν οι ακόλουθες παράμετροι: ο χώρος κατάστασης, η ενέργεια (στόχος) συνάρτηση $E()$, η υποψήφια διαδικασία γεννήτριας γείτονας $N()$, η συνάρτηση πιθανότητας αποδοχής $P()$, και η θερμοκρασία προγραμματισμού ανόπτησης $T()$ και αρχική θερμοκρασία. Αυτές οι επιλογές μπορούν να έχουν σημαντικό αντίκτυπο στην αποτελεσματικότητα της μεθόδου.

Κεφάλαιο 3

Αρχιτεκτονική

3.1 Τοπικός Ελεγκτής (LC)	18
3.2 Το σύστημα IMCF	19
3.3 Δεδομένα εισόδου	20
3.4 Ο αλγόριθμος IMCF	21
3.4.1 Σχέδιο απόσβεσης (AP)	22
3.4.2 Αρχιτεκτονική αλγόριθμου ενεργειακού σχεδίου (EP)	24
3.4.3 Αρχιτεκτονική αλγόριθμου Green Planner (GP)	26
3.5 Οι βασικές προσεγγίσεις	28
3.5.1 Βασική προσέγγιση χωρίς κανόνες (NR)	29
3.5.2 Βασική προσέγγιση με χρήση όλων των μέτα-κανόνων (MR)	29

Σε αυτό το κεφάλαιο αναλύεται η αρχιτεκτονική του συστήματος IMCF που υλοποιήθηκε το οποίο αποτελείται από 2 βασικά συστατικά τον τοπικό ελεγκτή (LC) και το IMCF συστατικό. Επίσης αναλύονται και τα δεδομένα εισόδου που χρησιμοποιούνται από τα συστατικά.

3.1 Τοπικός Ελεγκτής (LC)

Ο τοπικός ελεγκτής είναι ανεπτυγμένος στην γλώσσα προγραμματισμού JAVA και είναι σχεδιασμένος για εγκατάσταση σε μια μικροσυσκευή όπως το raspberry Pi, ο οποίος θα τρέχει στο τοπικό δίκτυο του χρήστη. Ο τοπικός ελεγκτής βρίσκεται σε άμεση

επικοινωνία με τις συσκευές του διαδικτύου των πραγμάτων για να τους δώσει εντολές βάσει τις προτιμήσεις του χρήστη ή της ομάδας χρηστών του συστήματος IMCF. Επίσης ο τοπικός ελεγκτής είναι ανεπτυγμένος ως επέκταση του ανοιχτού λογισμικού για έξυπνο σπίτι, εν ονόματι OpenHab, το οποίο είναι ένα σύστημα μέσα από το οποίο ένας χρήστης μπορεί να αλληλοεπιδράσει με τις διάφορες έξυπνες συσκευές στο περιβάλλον του. Το OpenHab μπορεί να εγκατασταθεί με εφαρμογή στο κινητό, ούτως ώστε ο έλεγχος να γίνεται μέσω κινητού για ευκολία.

Για την καλύτερη κατανόηση της λειτουργίας του τοπικού ελεγκτή μπορεί να θεωρηθεί το ακόλουθο παράδειγμα. Έστω ότι ένας χρήστης θέλει να μειώσει την θερμοκρασία ενός κλιματιστικού στο από 27 βαθμούς κελσίου σε 25. Για επίτευξη αυτής της μείωσης της θερμοκρασίας ο χρήστης πρέπει να αλληλοεπιδράσει μέσω τοπικού ελεγκτή, ο οποίος τελικά θα επικοινωνήσει με το κλιματιστικό.

Για την επικοινωνία με τον τοπικό ελεγκτή εκτός τοπικού δικτύου χρειάζεται η σύνδεση με τον Cloud Controller (CC) ο οποίος ελέγχει τον τοπικό ελεγκτή από απόσταση ή η σύνδεση με το τοπικό δίκτυο μέσω ενός VPN. Ο Cloud Controller δεν έχει υλοποιηθεί στους σκοπούς αυτής της διπλωματικής εργασίας.

3.2 Το σύστημα IMCF

Το συστατικό IMCF είναι μια επέκταση του λογισμικού του τοπικού ελεγκτή, το οποίο επιτρέπει την προσαρμογή των προτιμήσεων ευκολίας για την επίτευξη των μακροπρόθεσμων ενεργειακών στόχων του χρήστη ή της ομάδας χρηστών. Έχει αναπτυχθεί με τέτοιο τρόπο που ενσωματώνει την υλοποίηση του αλγορίθμου Energy Planner και Green Planner, αλλά και της γραφικής διαπροσωπείας του χρήστη και της αποθήκευσης των πληροφοριών, που απαιτούνται για μπορεί ο χρήστης να αλληλοεπιδράσει με το σύστημα. Ο αλγόριθμος EP εφαρμόζεται ως βιβλιοθήκη JAVA και λαμβάνει τις ρυθμίσεις των χρηστών από ένα MariaDB persistence layer. Το επίπεδο της αποθήκευσης κατά τη χρήση του συστήματος, το οποίο έχει διαμορφωθεί ούτως ώστε ο ορισμός μέτα-κανόνων στον πίνακα μέτα-κανόνων μέσω να γίνεται απρόσκοπτα μέσω της γραφικής διαπροσωπείας.

Η γραφική διαπροσωπεία είναι αναπτυγμένη ως μια εφαρμογή Laravel PHP ακολουθώντας το αρχιτεκτονικό μοτίβο μοντέλο-προβολή-ελεγκτή και με χρήση

JavaScript και HTML. Η βιβλιοθήκη JAVA αποτελείται από περίπου 4500 γραμμές κώδικα και η γραφική διαπροσωπεία από 3000. Η εκτέλεση του κώδικα της γραφικής διαπροσωπείας γίνεται μέσω του διακομιστή ιστού NGINX το οποίο είναι διαθέσιμο στο Raspberry PI, ενώ για την Energy Planner και Green Planner τρέχει κάθε 30 λεπτά αξιόπιστα μέσω cronjob daemon.

Το σύστημα IMCF προσφέρει τις ακόλουθες επιλογές σε περίπτωση που πρέπει να γίνει ενεργοποίηση/απενεργοποίηση κάποιων συσκευών:

- Binding mode: χρήση του πλούσιου οικοσυστήματος διαθέσιμων γεφυρών από το OpenHAB για αλληλεπίδραση με τοπικές συσκευές. Αυτή η επιλογή είναι προεπιλεγμένη για το σύστημα IMCF.
- Extended mode: το σύστημα εφαρμόζει τοπικά προσαρμοσμένες οδηγίες για την ενεργοποίηση και απενεργοποίηση των έξυπνων συσκευών.

Το σύστημα IMCF προστατεύεται μέσω της ταυτοποίησης που προσφέρει το OpenHab και Laravel. Επίσης ο τοπικός ελεγκτής βρίσκεται στο τοπικό δίκτυο του χρήστη και προστατεύεται από το firewall του λειτουργικού συστήματος Ubuntu. Συνεπώς, το σύστημα είναι αρκετά ασφαλές, καθώς κάποιος με κακόβουλες προθέσεις θα πρέπει να διαπεράσει την ασφάλεια που προσφέρει το OpenHab και η Laravel για να μπορέσει να κάνει αλλαγές ή να διαπεράσει το firewall.

3.3 Δεδομένα εισόδου

Η υλοποίηση των αλγορίθμων EP και GP χρειάζονται ως δεδομένα εισόδου τους κανόνες από τον πίνακα μέτα-κανόνων MRT και ένα προφίλ κατανάλωσης ενέργειας ECP. Για κάθε εκτέλεση των αλγορίθμων το σύστημα χρησιμοποιεί τους ανανεωμένους μέτα-κανόνες τους οποίους λαμβάνει με αίτημα από το API του συστατικού IMCF. Το προφίλ κατανάλωσης ενέργειας αποτελείται από την μηνιαία κατανάλωση ενέργειας του χρήστη σε kWh. Επίσης για τον αλγόριθμο GP χρειάζονται και προβλέψεις για τον καιρό. Ακολουθεί πίνακας με παράδειγμα για το ECP [11].

Μήνες	kWh τον μήνα	kWh την ώρα
Ιανουάριος	775.50	1.04
Φεβρουάριος	528.75	0.71
Μάρτιος	246.75	0.33
Απρίλιος	141.00	0.19
Μάιος	176.25	0.24
Ιούνιος	211.50	0.28
Ιούλιος	246.75	0.33
Αύγουστος	317.25	0.43
Σεπτέμβριος	211.50	0.28
Οκτώβριος	176.25	0.24
Νοέμβριος	211.50	0.28
Δεκέμβριος	423.00	0.57

Πίνακας 3.1 Παράδειγμα ECP [11]

3.4 Ο αλγόριθμος IMCF

Ο αλγόριθμος IMCF [11] αποτελείται από δύο υπορουτίνες, το σχέδιο απόσβεσης (AP) και το Ενεργειακό Σχέδιο (EP) ή τον αλγόριθμο Green Planner (GP). Το σχέδιο απόσβεσης είναι υπεύθυνο για τον υπολογισμό του μέγιστου περιορισμού του ενεργειακού προϋπολογισμού μέσω ενός προεπιλεγμένου τύπου απόσβεσης. Στη συνέχεια εκτελείται μια προσέγγιση τεχνητής νοημοσύνης κάθε t δευτερόλεπτα για μια χρονική περίοδο p για τη δημιουργία μιας λύσης ενεργειακού σχεδίου s^* για τη βελτιστοποίηση του σφάλματος άνεσης.

$$\min_{FCE} = \sum_{k=1}^t \left(\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^D ce_j(MR_i) \right)$$

Εξίσωση 1

Όπου το ce_j είναι μεταξύ της επιθυμητής τιμής εξόδου $\Omega_i^j \in R$ ενός κανόνα που ορίζεται από έναν χρήστη όπως τη θερμοκρασία ή την ένταση φωτός και την πραγματική τιμή $O_i^j \in R$ ορισμένη από τον ελεγκτή, δηλαδή $ce = |\Omega_i^j| - |O_i^j|$.

$$F_E = \sum_{k=1}^t \left(\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^D e_j(MR_i) \right)$$

Εξίσωση 2

Η κατανάλωση ενέργειας F_E πρέπει να είναι μικρότερη από τον συνολικό διαθέσιμο προϋπολογισμό ενέργειας E_p για το χρονικό διάστημα p κατά το οποίο εκτελείται ο αλγόριθμος. Στην εξίσωση 2 το N είναι αριθμός των μέτα-κανόνων, D είναι ο αριθμός των συσκευών διαδικτύου των πραγμάτων και το e_j είναι η κατανάλωση ενέργειας μιας συσκευής j δοσμένη από τη παραγωγή O_i^j του μέτα-κανόνα MR_i από:

$$E_j = \begin{cases} e_j, & O_i^j \text{ is executed} \\ 0, & \text{otherwise} \end{cases}$$

Γ_j είναι η εκπομπή CO_2 που παράγεται από την ενεργοποίηση της συσκευής j και για τον υπολογισμό της χρησιμοποιείται η ενεργειακή κατανάλωση E της συσκευής και την ένταση εκπομπής γ CO_2 της χώρας.

$$\Gamma_j = \begin{cases} E_j, & \text{if sunlight} \\ 0, & \text{otherwise} \end{cases}$$

Υπόκειται στην ικανοποίηση του $F_r \leq \Gamma_p$ όπου,

$$F_r = \sum_{k=1}^t \left(\frac{1}{N} \sum_{i=1}^N \sum_{j=1}^D \Gamma_j(E_j(MR_i, \gamma)) \right)$$

Εξίσωση 3

Η εκπομπή διοξειδίου του άνθρακα F_r πρέπει να είναι μικρότερη από την μέγιστη επιθυμητή εκπομπή CO_2 Γ_p , για ολόκληρη την περίοδο p κατά την οποία πραγματοποιείται η εκτέλεση του αλγορίθμου.

3.4.1 Σχέδιο απόσβεσης (AP)

Αρχικά εκτελείται το σχέδιο απόσβεσης AP [11] το οποίο είναι μια υπορουτίνα η οποία υπολογίζει τον περιορισμό του ενεργειακού προϋπολογισμού E_p , με χρήση του προφίλ κατανάλωσης ενέργειας ECP. Κάποιες στρατηγικές απόσβεσης έχουν ως ακολούθως:

- i. Στρατηγική γραμμικής απόσβεσης (LAF), όπου η συνολική κατανάλωση ενέργειας TE μπορεί να κατανεμηθεί γραμμικά σε μια προκαθορισμένη χρονική περίοδο p , με διάρκεια t , η οποία μπορεί να οριστεί ως ετήσια, μηνιαία, καθημερινή, ωριαία και ούτω καθεξής, δίνοντας τον περιορισμό του ενεργειακού προϋπολογισμού:

$$E_p = \frac{TE}{t}$$

- ii. Στρατηγική γραμμικής απόσβεσης μπαλονιών (BLAF), όπου ο χρήστης εξοικονομεί ένα ποσοστό π από τη συνολική κατανάλωση ενέργειας TE για ένα χρονικό διάστημα $\lambda < t$, το λεγόμενο μπαλόνι σ , το οποίο χρησιμοποιείται στην υπόλοιπη χρονική περίοδο $\lambda' = t - \lambda$ όπου η κατανάλωση ενέργειας είναι υψηλότερη. Ο περιορισμός ενεργειακού προϋπολογισμού E_p για μια περίοδο p διάρκειας t υπολογίζεται ως εξής:

$$E_p = \begin{cases} \frac{TE}{t} - \frac{\sigma}{\lambda}, & \text{for } \lambda \text{ period} \\ \frac{TE}{t} + \frac{\sigma}{\lambda}, & \text{for } \lambda' \text{ period} \end{cases}$$

- iii. Στρατηγική απόσβεσης με βάση το ECP (EAF), όπου υπολογίζεται ένα σύνολο βαρών με χρήση του διανύσματος ECP, τα οποία χρησιμοποιούνται για τον καθορισμό των περιορισμών του ενεργειακού προϋπολογισμού για μια περίοδο που καθορίζεται από τον χρήστη σε ένα διαθέσιμο ενεργειακό προϋπολογισμό E :

$$E_p = \left\{ \frac{w_i \times E}{t/|ECP|} \right\}, \text{ for } i = 1, \dots, |ECP|,$$

$$\text{where } w_i = \frac{TE}{ECP_i} \text{ and } \sum_{i=1}^{|ECP|} w_i = 1$$

Το TE είναι η συνολική κατανάλωση ενέργειας που προέρχεται από το ECP, E είναι ο διαθέσιμος προϋπολογισμός ενέργειας που καθορίζεται από το χρήστη, $|ECP|$ είναι το μέγεθος του διανύσματος του προφίλ κατανάλωσης ενέργειας και $t/|ECP|$ ομαλοποιεί τον ενεργειακό προϋπολογισμό βάσει της χρονικής διάρκειας κοκκοποίησης t .

Για όλες τις στρατηγικές απόσβεσης η μέγιστη εκπομπή CO_2 Γ_p υπολογίζεται ως

$$\Gamma_p = E_p \times \gamma$$

Και ο περιορισμός F_Γ υπολογίζεται απευθείας από την Εξίσωση 3.

3.4.2 Αρχιτεκτονική αλγόριθμου ενεργειακού σχεδίου (EP)

Σε αυτή την υποενότητα αναλύεται ο αλγόριθμος EP και τα σχετικά στοιχεία και παραμέτρους του.

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 $s^* = (s_1, \dots, s_N)$

```

1: AP(apl, p, ECP)           ▷ Amortization Plan Routine
2:   switch (apl)
3:     a:  $E_p \leftarrow LAF(t, ECP)$            ▷ use linear Eq. (3)
4:     b:  $E_p \leftarrow BLAF(t, ECP)$          ▷ use balloon Eq. (4)
5:     c:  $E_p \leftarrow EAF(t, ECP)$          ▷ use ECP-based Eq. (5)
6:
7:   EP(MRT, k,  $\tau_{max}$ , i,  $E_p$ )           ▷ Energy Plan Routine
8:      $s^* \leftarrow init_i(MRT)$            ▷  $s^*$ : initial solution for time i
9:      $(F_E, F_{CE}) \leftarrow evaluate(s^*)$    ▷ with Equations (1),(2)
10:    While  $\tau < \tau_{max}$  do                 ▷  $\tau$ : current iteration
11:       $s \leftarrow optimization(s^*)$        ▷ randomly select k
        positions and swap their binary value
12:       $(F_E, F_{CE}) \leftarrow evaluate(s)$    ▷ with Equations (1),(2)
13:      If  $(F_E(s) \leq E_p) \ \&\& \ (F_{CE}(s) < F_C(s^*))$  then
14:         $s^* \leftarrow s$                    ▷ Set s as the current solution  $s^*$ 
15:      EndIf
16:       $\tau ++$                                ▷ Increase iterations
17:    EndWhile
18:    return  $s^*$                              ▷ Return the final energy plan solution
19:
20: apl  $\leftarrow AP(apl, t, ECP)$ ;
21: return  $(\forall_i^t EP(MRT, k, \tau_{max}, i, apl))$ 

```

Σχήμα 3.1 Ο αλγόριθμος IMCF με χρήση της EP [11]

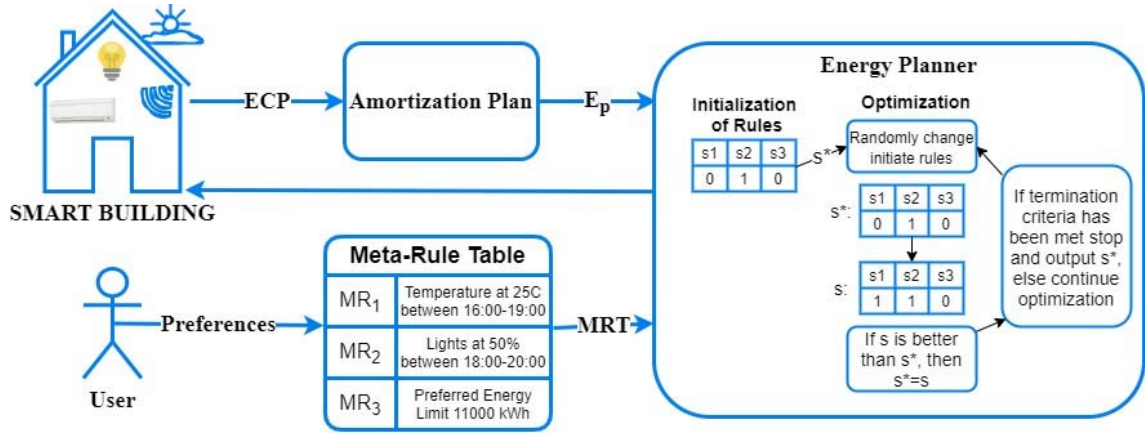
Μια λύση ενεργειακού σχεδίου είναι ένα διάνυσμα $s = \{s_1, s_2, s_3, \dots, s_N\}$, όπου το N είναι ο αριθμός των μέτα-κανόνων. Ένα στοιχείο s_i αντιπροσωπεύει έναν μέτα-κανόνα στον πίνακα MRT, όπου $s_i = 0$ σημαίνει αγνόηση του μέτα-κανόνα *i* του πίνακα MRT και $s_i = 1$ σημαίνει χρήση του μέτα-κανόνα *i*.

Στην αρχή της τοπικής ευρετικής αναζήτησης αναπτύσσεται μια αρχική λύση s^* που καθορίζει την αρχική κατάσταση του αλγορίθμου. Μια αρχική λύση μπορεί να δημιουργηθεί τυχαία ή ντετερμινιστικά. Μια ντετερμινιστική λύση για το EP είναι να οριστούν όλοι οι κανόνες s_i σε 1, πράγμα που σημαίνει ότι όλοι οι μέτα-κανόνες θα χρησιμοποιηθούν για την ανάπτυξη της λύσης s^* , ευνοώντας με αυτόν τον τρόπο το σφάλμα ευκολίας, αλλά με μεγάλη πιθανότητα παραβίασης του περιορισμού E_P . Στην περίπτωση τυχαίας αρχικοποίησης, οι τιμές όλων των στοιχείων του διανύσματος επιλέγονται τυχαία.

Η βελτιστοποίηση επιτυγχάνεται με ευρετική τοπική αναζήτηση αναρρίχησης λόφου όπου χρησιμοποιείται για τοπική βελτιστοποίηση με γειτονιές που περιλαμβάνουν αλλαγή έως και k συστατικών της λύσης, η οποία αναφέρεται και ως k -opt. Κατά τη διαδικασία βελτιστοποίησης k , οι μέτα-κανόνες επιλέγονται τυχαία από το διάνυσμα s εάν θα αγνοηθούν ή θα χρησιμοποιηθούν.

Κάθε λύση s αξιολογείται χρησιμοποιώντας τις μετρήσεις απόδοσης F_E και F_{CE} των Εξισώσεων Εξίσωση 1 και Εξίσωση 2. Μια λύση s θεωρείται καλύτερη και αντικαθιστά την τρέχουσα καλύτερη λύση s^* εάν η κατανάλωση ενέργειας F_E είναι μικρότερη από τον συνολικό διαθέσιμο προϋπολογισμό ενέργειας E_P και αν το συνολικό ποσοστό λάθους για τη λύση s είναι μικρότερο από τη λύση s^* .

Ο αλγόριθμος ενεργειακού σχεδίου τερματίζει όταν φτάσει στις t_{max} επαναλήψεις. Εναλλακτικά, συνεχίζει μέχρι να μην υπάρχουν καλύτερες λύσεις. Σε περίπτωση που υπάρχουν ελλείψεις στη γνώση σχετικά με τη βέλτιστη λύση, ο αλγόριθμος μπορεί να μπει σε ατέρμονο βρόγχο.



Σχήμα 3.2 Παράδειγμα εκτέλεσης του IMCF με EP

3.4.3 Αρχιτεκτονική αλγόριθμου Green Planner (GP)

Σε αυτή την υποενότητα αναλύεται ο αλγόριθμος GP [21] και τα σχετικά στοιχεία και παραμέτρους του.

Algorithm 1 IMCF: generates a green-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; T : Temperature
Output: An green plan solution $s^* = (s_1, \dots, s_N)$

```

1: AP( $apl, p, ECP$ )                                ▶ Amortization Plan Routine
2: switch ( $apl$ )
3:   a:  $E_p \leftarrow LAF(t, ECP)$                     ▶ use linear Eq. (4)
4:   b:  $E_p \leftarrow BLAF(t, ECP)$                   ▶ use balloon Eq. (5)
5:   c:  $E_p \leftarrow EAF(t, ECP)$                   ▶ use ECP-based Eq. (6)
6:   return  $\Gamma_p = E_p \times \gamma$                         ▶ max  $CO_2$  emission for  $\gamma$   $CO_2$  emission intensity
7:
8: GP(MRT,  $k, \tau_{max}, i, \Gamma_p$ )                      ▶ Green Plan Routine
9:    $s^* \leftarrow init_i(MRT)$                         ▶  $s^*$ : initial solution for time  $i$ 
10:  ( $F_{CE}, F_E, F_T$ )  $\leftarrow evaluate(s^*)$           ▶ with Equations (1),(2),(3)
11:  While  $\tau < \tau_{max}$  do                             ▶  $\tau$ : current iteration
12:     $s \leftarrow optimization(s^*)$                 ▶ randomly select  $k$  positions and swap their binary value
13:    ( $F_{CE}, F_E, F_T$ )  $\leftarrow evaluate(s)$           ▶ with Equations (1),(2),(3)
14:    If ( $F_E(s) \leq E_p$ ) && ( $F_{CE}(s) < F_{CE}(s^*)$ ) && ( $F_T(s) < \Gamma_p$ ) then
15:       $s^* \leftarrow s$                                 ▶ Set  $s$  as the current solution  $s^*$ 
16:    Else If  $rand(0, 1) \leq P(F_{CE}(s^*), F_{CE}(s), T)$  && ( $F_{CE}(s) < F_{CE}(s^*)$ ) && ( $F_T(s) < \Gamma_p$ ) then
17:       $s^* \leftarrow s$                                 ▶ Set  $s$  as the current solution  $s^*$ 
18:    EndIf
19:     $\tau++$                                              ▶ Increase iterations
20:  EndWhile
21:  return  $s^*$                                           ▶ Return the final green plan solution
22:
23:  $\Gamma_p \leftarrow AP(apl, t, ECP)$ ;
24: return ( $\forall_i GP(MRT, k, \tau_{max}, i, \Gamma_p)$ )

```

Σχήμα 3.3 Ο αλγόριθμος IMCF με χρήση της GP [21]

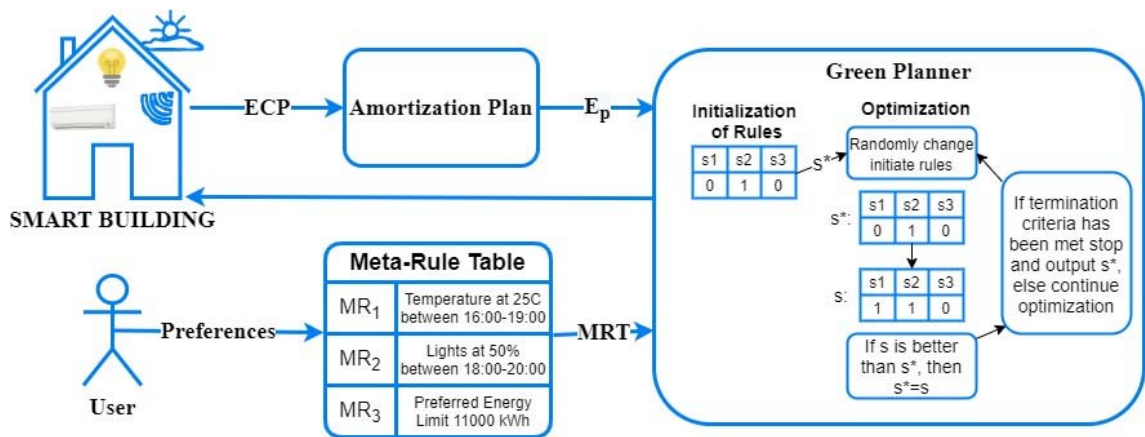
Μια πράσινη λύση ενεργειακού σχεδίου είναι ένα διάνυσμα $s = \{s_1, s_2, s_3, \dots, s_N\}$ όπου το N είναι ο αριθμός των μέτα-κανόνων. Ένα στοιχείο s_i αντιπροσωπεύει έναν μέτα-κανόνα στον πίνακα MRT, όπου $s_i = 0$ σημαίνει αγνόηση του μέτα-κανόνα i του πίνακα MRT και $s_i = 1$ σημαίνει χρήση του μέτα-κανόνα i .

Στην αρχή της αναπτύσσεται μια αρχική λύση s^* που καθορίζει την αρχική κατάσταση του αλγορίθμου. Μια ντετερμινιστική λύση για το GP είναι να οριστούν όλοι οι κανόνες s_i σε 1, πράγμα που σημαίνει ότι όλοι οι μέτα-κανόνες θα χρησιμοποιηθούν για την ανάπτυξη της λύσης s^* , ευνοώντας με αυτόν τον τρόπο το σφάλμα ευκολίας, αλλά με μεγάλη πιθανότητα παραβίασης του περιορισμού Γ_p . Στην περίπτωση τυχαίας αρχικοποίησης, οι τιμές όλων των στοιχείων του διανύσματος επιλέγονται τυχαία.

Για το βήμα βελτιστοποίησης, χρησιμοποιείται μια ευρετική προσομοιωμένης ανόπτησης για τη μετατροπή της λύσης της τρέχουσας κατάστασης s^* σε λύση μιας νέας κατάστασης s επιλέγοντας τυχαία και ανταλλάσσοντας έως k στοιχεία του s^* . Η πιθανότητα πραγματοποίησης της μετάβασης από την τρέχουσα κατάσταση s^* σε μια υποψήφια νέα κατάσταση s καθορίζεται από μια συνάρτηση πιθανότητας αποδοχής $P(e(s^*), e(s), T)$, όπου $e(s^*)$ και $e(s)$ είναι η ενέργεια (ή αξιολόγηση καταλληλότητας) των λύσεων που παράγονται σε καταστάσεις s^* και s , αντίστοιχα, και T είναι μια παγκόσμια παράμετρος που ποικίλλει από το χρόνο, που ονομάζεται θερμοκρασία. Η θερμοκρασία T παίζει καθοριστικό ρόλο στον έλεγχο της εξέλιξης της κατάστασης s , δεδομένου ότι μια μεγάλη T ευνοεί τις χοντρές παραλλαγές μεταξύ των καταστάσεων, όπου μια μικρή T ευνοεί περισσότερες παραλλαγές λεπτών κόκκων. Η παράμετρος T μπορεί να ρυθμιστεί είτε μέσω ενός προκαθορισμένου χρονοδιαγράμματος ανόπτησης είτε να υπολογιστεί σε σχέση με τον αριθμό των επαναλήψεων που έχουν περάσει (δηλαδή, $T = \frac{\tau+1}{\tau_{max}}$). Σε κάθε περίπτωση, απαιτείται σταδιακή μείωση του T καθώς προχωρά η προσομοίωση. Μια μετάβαση γίνεται πάντα αποδεκτή όταν το s είναι καλύτερο από το s^* και είναι πιθανώς αποδεκτό, με πιθανότητα $P(e(s^*), e(s), T)$ όταν s είναι χειρότερο από το s^* , επιτρέποντας με αυτόν τον τρόπο την ευρετική αποφυγή τοπικών βέλτιστων.

Κάθε λύση s αξιολογείται χρησιμοποιώντας τις μετρήσεις απόδοσης F_{CE} , F_E και F_T από τις εξισώσεις 1, 2, 3 αντίστοιχα. Μια λύση s θεωρείται καλύτερη και αντικαθιστά την τρέχουσα καλύτερη λύση s^* εάν η κατανάλωση ενέργειας F_E είναι μικρότερη από τον συνολικό διαθέσιμο προϋπολογισμό ενέργειας E_p , αν το συνολικό ποσοστό λάθους ευκολίας για τη λύση s είναι μικρότερο από τη λύση s^* και εάν η εκπομπή διοξειδίου του άνθρακα F_T είναι μικρότερη από την μέγιστη επιθυμητή εκπομπή Γ_p . Διαφορετικά, εάν το s είναι χειρότερο από το s^* , η s αντικαθιστά την s^* αν μια τυχαία τιμή $rand$ στο διάστημα $[0,1]$ είναι μικρότερη από την πιθανότητα $P(e(s^*), e(s), T)$.

Ο αλγόριθμος πράσινου σχεδίου τερματίζει όταν φτάσει στις τ_{max} επαναλήψεις. Εναλλακτικά, συνεχίζει μέχρι να μην υπάρχουν καλύτερες λύσεις. Σε περίπτωση που υπάρχουν ελλείψεις στη γνώση σχετικά με τη βέλτιστη λύση, ο αλγόριθμος μπορεί να μπει σε ατέρμονο βρόγχο.



Σχήμα 3.4 Παράδειγμα εκτέλεσης του IMCF με GP

3.5 Οι βασικές προσεγγίσεις

Για σκοπούς αποτίμησης των αλγορίθμων δημιουργήθηκαν κάποιες βασικές προσεγγίσεις οι οποίες αναφέρονται σε αυτή την υποενότητα.

3.5.1 Βασική προσέγγιση χωρίς κανόνες (NR)

Είναι η πρώτη βασική προσέγγιση, όπου αγνοεί όλους τους μέτα-κανόνες του πίνακα μέτα-κανόνων και δεν τροποποιεί τη συμπεριφορά των αυτόνομων συσκευών, επομένως αυτό έρχεται σε αντίθεση με την ευκολία του χρήστη. Η συνολική κατανάλωση ενέργειας F_E είναι προφανώς πάντα 0, δεδομένου ότι καμία συσκευή διαδικτύου των πραγμάτων δεν είναι ενεργοποιημένη. Από την άλλη, το F_{CE} μετριέται ως ποσοστό σφάλματος ευκολίας που θα είχε ένας χρήστης εάν αυτός ο χρήστης εκτέλεσε όλους τους κανόνες και το F_T είναι μόνο το κόστος της πραγματοποίησης αυτού του υπολογισμού. Κατά συνέπεια, η κατανάλωση ενέργειας αυτής της προσέγγισης είναι ελάχιστη και το σφάλμα ευκολίας είναι το μέγιστο.

3.5.2 Βασική προσέγγιση με χρήση όλων των μέτα-κανόνων (MR)

Είναι η δεύτερη βασική προσέγγιση, όπου χρησιμοποιεί όλους τους μέτα-κανόνες του πίνακα μέτα-κανόνων και ευνοεί την ευκολία του χρήστη. Η συνολική κατανάλωση ενέργειας F_E είναι μέγιστη σε αυτή την περίπτωση ενώ το ποσοστό σφάλματος F_{CE} είναι ελάχιστο.

Κεφάλαιο 4

Υλοποίηση Αλγορίθμων

4.1 Εισαγωγή dataset	30
4.2 Ενημέρωση των μέτα-κανόνων από το σύστημα IMCF.....	31
4.3 Εισαγωγή των μέτα-κανόνων στο σύστημα	32
4.4 Υλοποίηση αλγόριθμου noRule / noIFTTT	33
4.5 Υλοποίηση του αλγόριθμου Energy Planner.....	34
4.6 Υλοποίηση του αλγόριθμου Green Planner	37

Σε αυτό το κεφάλαιο αναλύεται η υλοποίηση των διάφορων αλγορίθμων σε βιβλιοθήκη JAVA. Η βιβλιοθήκη αποτελείται από τις κλάσεις `importData` όπου χρησιμοποιείται για την εισαγωγή των απαραίτητων δεδομένων για εκτέλεση των αλγορίθμων, την κλάση `noIFTTT`, την κλάση `EnergyPlanner` στην οποία υλοποιήθηκε ο αλγόριθμος EP και την κλάση `GreenPlanner` στην οποία υλοποιήθηκε ο αλγόριθμος GP.

4.1 Εισαγωγή dataset

Το σύνολο δεδομένων για διαμέρισμα ή κατοικία ή για φοιτητικές εστίες εισάγεται στο σύστημα μέσω ανάγνωσης του αρχείου και τοποθετείτε σε ένα `ArrayList` το οποίο χρησιμοποιείται από τους αλγόριθμους.

```

1. while (inputFile.hasNextLine()) {
2.     line = inputFile.nextLine();
3.     String temp[] = new String[7];
4.
5.     if (counter > 1) {
6.         String[] col = line.split("\\|");
7.         temp[0] = col[0].trim();
8.         temp[1] = col[1].trim();
9.         temp[2] = col[2].trim();
10.        temp[3] = col[3].trim();
11.        temp[4] = col[4].trim();
12.        temp[5] = col[5].trim();
13.        temp[6] = col[6].trim();
14.        // fills a list with all the input IoT data
15.        IotData.add(temp);
16.    }
17.
18.    counter++;
19. } // end of while loop

```

Μέσω του While loop διαβάζονται όλες οι γραμμές του αρχείου και χωρίζοντας κάθε γραμμή σε string token όπως ενδείκνυται στην εντολή της γραμμής 6, αναθέτονται σε ένα προσωρινό πίνακα String όπου στη συνέχεια τοποθετείται στο ArrayList στην γραμμή 15. Προχωρώντας στην επόμενη καταχώρηση του συνόλου δεδομένων έχουμε ένα μετρητή με όνομα counter ο οποίος αυξάνεται σταδιακά.

4.2 Ενημέρωση των μέτα-κανόνων από το σύστημα IMCF

Οι μέτα-κανόνες ανανεώνονται πριν κάθε εκτέλεση και για να γίνει αυτό κατεβάζοντας τους από το σύστημα IMCF με χρήση API σε μορφή JSON και μετά από απαραίτητη επεξεργασία γράφονται στο αρχείο μέτα-κανόνων.

```
1. String urlString = "http://10.16.30.73/api/get-metarules";
2. StringBuilder result = new StringBuilder();
3. URL url = new URL(urlString);
4. HttpURLConnection conn = url.openConnection();
5. conn.setRequestMethod("GET");
6. conn.connect();
7. int responsecode = conn.getResponseCode();
8.
9. if (responsecode == 200) {
10.     BufferedReader br = new BufferedReader(conn);
11.     String line;
12.     while ((line = br.readLine()) != null) {
13.         result.append(line);
14.     }
15. }
```

Στη γραμμή 1 δηλώνεται ο σύνδεσμος για το API όπου στη συνέχεια με αίτημα GET στο API λαμβάνεται ο πίνακας μέτα-κανόνων και διαβάζεται από το While loop στη γραμμή 12-14. Έπειτα γίνεται επεξεργασία του πίνακα ο οποίος λήφθηκε σε μορφή JSON και αποθηκεύεται στο αρχείο των μέτα-κανόνων. Οι μέτα-κανόνες λαμβάνονται από τη τοπική βάση δεδομένων του πλαισίου μας MariaDB.

4.3 Εισαγωγή των μέτα-κανόνων στο σύστημα

Η εισαγωγή των μέτα-κανόνων γίνεται με την ανάγνωση του αρχείου μέτα-κανόνων αφού ενημερωθεί όπως έχει αναφερθεί πιο πάνω. Οι κανόνες μπαίνουν σε ένα ArrayList για χρήση τους από τους αλγορίθμους.

```

1. File file = new File(pathMRrules);
2. // loop through every line
3. while (inputFile.hasNextLine()) {
4.     line = inputFile.nextLine();
5.     String[] col = line.split("\\|");
6.     String temp[] = new String[4];
7.
8.     if (counter > 1) {
9.         temp[0] = col[0].trim();
10.        temp[1] = col[2].trim();
11.        temp[2] = col[3].trim();
12.        temp[3] = col[1].trim();
13.        MRrules.add(temp);
14.    }
15.    counter++;
16. } // end of while loop

```

Αφού ανοιχτεί το αρχείο διαβάζεται κάθε γραμμή από το while loop (γραμμή 3-12) και αφού κατακερματιστεί κάθε γραμμή, τα δεδομένα κάθε γραμμής αποθηκεύονται στο ArrayList στην γραμμή 12, έτσι ώστε να χρησιμοποιηθούν για επεξεργασία από το πλαίσιο μας.

4.4 Υλοποίηση αλγόριθμου noRule / noIFTTT

Ο αλγόριθμος no-Rule υπολογίζει το μέγιστο (F_{CE}). Ο αλγόριθμος εκτελείται πάντα πριν τον αλγόριθμο EP και GP γιατί γίνεται χρήση του μέγιστου F_{CE} από αυτούς. Αυτό γίνεται με χρήση ενός συνόλου δεδομένων για διαμέρισμα ή κατοικία ή για φοιτητικές εστίες και τη χρήση των μέτα-κανόνων. Για κάθε μέτρηση από το σύνολο δεδομένων ελέγχονται όλοι οι μέτα-κανόνες, ούτως ώστε να μετρηθεί το σφάλμα λάθους σε περίπτωση παραβίασης κάποιου από αυτούς.

Οι έλεγχοι έχουν ως ακολούθως:

Αρχικά ελέγχεται αν η μέτρηση περιλαμβάνει θερμοκρασία ή ένταση φωτός.

```
// gets T for temperatures
if (dataSplitted[2].startsWith("T")) {
// gets LS for light sensor
if (dataSplitted[2].startsWith("LS")) {
```

Εάν γίνεται χρήση των μέτα-κανόνων τύπου IFTTT χρησιμοποιείται ο μήνας στον οποίο λήφθηκε η μέτρηση για να βρούμε σε ποια εποχή λήφθηκαν τα δεδομένα.

```
int iSeason = Integer.parseInt(dataSplitted[0]);
```

Στη συνέχεια γίνεται έλεγχος σε ποια εποχή βρισκόμαστε, δηλαδή αν ο μήνας που λήφθηκαν τα δεδομένα είναι χειμώνας ή καλοκαίρι.

```
// checks if SUMMER
if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
// checks if WINTER
else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
```

Αν δεν ισχύει η προτιμώμενη ένταση φωτός ή η προτιμώμενη θερμοκρασία για την ώρα που έχει τεθεί ή για την εποχή που έχει τεθεί τότε αυξάνεται το συνολικό λάθος ευκολίας με τη διαφορά των ποσών από τη μέτρηση και τον μέτα-κανόνα.

4.5 Υλοποίηση του αλγόριθμου Energy Planner

Ο αλγόριθμος EP χρησιμοποιεί το σύνολο δεδομένων για διαμέρισμα ή κατοικία ή για φοιτητικές εστίες, τον πίνακα μέτα-κανόνων και το μέγιστο λάθος που υπολογίζεται από τον αλγόριθμο No-Rule. Ο αλγόριθμος υπολογίζει το σφάλμα ευκολίας (F_{CE}) και την κατανάλωση ενέργειας (F_E).


```

1.  for (int i = 0; i < iotdata.size(); i++) {
2.
3.      String[] dataSplitted = iotdata.get(i);
4.      double tempHourChange = getHour(dataSplitted[0]);
5.
6.      if (hourChange == tempHourChange) {
7.
8.          if (i == iotdata.size() - 1) {
9.              recordsHourly.add(iotdata.get(i));
10.             double hourly_ms = calculateKWH(recordsHourly);
11.             total_ms = total_ms + hourly_ms;
12.             recordsHourly.clear();
13.         } else {
14.             recordsHourly.add(iotdata.get(i));
15.         }
16.
17.     } else {
18.         hourChange = tempHourChange;
19.         hourly_ms = calculateKWH(recordsHourly);
20.         total_ms = total_ms + hourly_ms;
21.         recordsHourly.clear();
22.         recordsHourly.add(iotdata.get(i));
23.     }
24.
25. }

```

Για κάθε καταγραφή από το σύνολο δεδομένων βλέπουμε τότε αλλάζει η ώρα και υπολογίζεται η καλύτερη λύση με χρήση του Amortization Plan. Για κάθε ώρα υπολογίζεται η κατανάλωση ενέργειας και το σφάλμα λάθους για την καλύτερη λύση με χρήση της μεθόδου calculateKW, η οποία είναι εμπνευσμένη από την αναρρίχηση λόφου (hill climbing). Καλύτερη λύση αποτελεί το διάνυσμα των ενεργοποιημένων και απενεργοποιημένων κανόνων, για το οποίο το σφάλμα ευκολίας και η κατανάλωση, είναι ελάχιστα. Η calculateKW είναι recursion μέθοδος όπου εκτελείται για ένα συγκεκριμένο αριθμό επαναλήψεων για κάθε καταγραφή του συνόλου. Σε κάθε πρώτη εκτέλεση της μεθόδου γίνεται τυχαία επιλογή της ενεργοποίησης και απενεργοποίησης των κανόνων όπως έχει ως ακολούθως. Στην εντολή στη γραμμή 4 επιλέγεται τυχαία ένας αριθμός από το 0 μέχρι το 1. Αν είναι 0 τότε ο μέτα-κανόνας θα είναι απενεργοποιημένος και η αντίστοιχη θέση του στον πίνακα με όνομα randomRule είναι false, όπως φαίνεται στην γραμμή εντολής 7 και προσθέτετε σαν απενεργοποιημένος κανόνας στην λίστα με όνομα removedMetaRules, όπως φαίνεται στην γραμμή εντολής 8. Διαφορετικά ο μέτα-κανόνας θα είναι ενεργοποιημένος και η αντίστοιχη θέση του στον πίνακα με όνομα randomRule είναι true όπως φαίνεται στην γραμμή εντολής 10.

```

1. for (int d = 0; d < randomRule.length; d++) {
2.
3.     // generates random number from 0 to 1
4.     random = rand.nextInt(2);
5.
6.     if (random == 0) {
7.         randomRule[d] = false;
8.         removedMetaRules.add(d);
9.     } else {
10.        randomRule[d] = true;
11.    }
12.
13. }

```

Στις υπόλοιπες επαναλήψεις γίνεται τυχαία επιλογή της ενεργοποίησης και απενεργοποίησης μέχρι k κανόνων το οποίο αντιπροσωπεύετε από τη μεταβλητή numberOfLoops. Στην γραμμή εντολής 4 παρακάτω, παράγεται τυχαία ένας αριθμός από το 0 μέχρι τον αριθμό των μέτα-κανόνων, όπου αντιπροσωπεύει την αλλαγή της κατάστασης του συγκεκριμένου κανόνα. Αν είναι ενεργοποιημένος τότε η θέση του στον πίνακα randomRule από true γίνεται false και αφαιρείται από την λίστα απενεργοποιημένων κανόνων όπως φαίνεται από τις γραμμές εντολών 7-8. Διαφορετικά, ο κανόνας ενεργοποιείται και αφαιρείται από τη λίστα με τους απενεργοποιημένους κανόνες, όπως φαίνεται από τις γραμμές εντολών 10-11.

```

1. for (int i = 0; i < numberOfLoops; i++) {
2.
3.     // random here is the rule to be removed
4.     random = rand.nextInt(numrules);
5.
6.     if (randomRule[random]) {
7.         randomRule[random] = false;
8.         removedMetaRules.add(random);
9.     } else {
10.        randomRule[random] = true;
11.        removedMetaRules.remove(random);
12.    }
13.
14. }

```

4.6 Υλοποίηση του αλγόριθμου Green Planner

Ο αλγόριθμος GP χρησιμοποιεί το σύνολο δεδομένων για διαμέρισμα ή κατοικία ή για φοιτητικές εστίες, τον πίνακα μέτα-κανόνων και το μέγιστο λάθος που υπολογίζεται από τον αλγόριθμο No-Rule. Ο αλγόριθμος υπολογίζει το σφάλμα ευκολίας (F_C) την κατανάλωση ενέργειας (F_E) και τις εκπομπές CO_2 (F_T).

```
1. for (int i = 0; i < iotdata.size(); i++) {
2.
3.     String[] dataSplitted = iotdata.get(i);
4.     double tempDayChange = getDay(dataSplitted[0]);
5.
6.     if (dayChange == tempDayChange) {
7.
8.         if (i == iotdata.size() - 1) {
9.             recordsDaily.add(iotdata.get(i));
10.            double hourly_ms = calculateKWH(recordsHourly);
11.            total_ms = total_ms + hourly_ms;
12.            recordsDaily.clear();
13.        } else {
14.            recordsDaily.add(iotdata.get(i));
15.        }
16.
17.    } else {
18.        dayChange = tempDayChange;
19.        hourly_ms = calculateKWH(recordsDaily);
20.        total_ms = total_ms + hourly_ms;
21.        recordsDaily.clear();
22.        recordsDaily.add(iotdata.get(i));
23.    }
24.
25. }
```

Για κάθε καταγραφή από το σύνολο δεδομένων βλέπουμε πότε αλλάζει η μέρα και υπολογίζεται η καλύτερη λύση ανά ώρα, με χρήση του Amortization Plan. Για κάθε μέρα υπολογίζεται η κατανάλωση ενέργειας, το σφάλμα λάθους και η εκπομπή διοξειδίου του άνθρακα για την καλύτερη ολική λύση με χρήση της μεθόδου calculateKW, η οποία είναι εμπνευσμένη από την προσομοιωμένη απόσπηση (simulated annealing). Καλύτερη λύση αποτελεί το διάνυσμα των ενεργοποιημένων και απενεργοποιημένων κανόνων, για το οποίο το σφάλμα ευκολίας, η κατανάλωση ενέργειας και η εκπομπή διοξειδίου του άνθρακα, είναι ελάχιστα. Η calculateKW είναι recursion μέθοδος όπου εκτελείται για ένα

συγκεκριμένο αριθμό επαναλήψεων για κάθε καταγραφή. Σε κάθε πρώτη εκτέλεση της μεθόδου γίνεται τυχαία επιλογή της ενεργοποίησης και απενεργοποίησης των κανόνων όπως έχει ως ακολούθως. Στην εντολή στη γραμμή 4 επιλέγεται τυχαία ένας αριθμός από το 0 μέχρι το 1. Αν είναι 0 τότε ο μέτα-κανόνας θα είναι απενεργοποιημένος και η αντίστοιχη θέση του στον πίνακα με όνομα `randomRule` είναι `false`, όπως φαίνεται στην γραμμή εντολής 7 και προσθέτετε σαν απενεργοποιημένους κανόνες στην λίστα με όνομα `removedMetaRules`, όπως φαίνεται στην γραμμή εντολής 8. Διαφορετικά ο μέτα-κανόνας θα είναι ενεργοποιημένος και η αντίστοιχη θέση του στον πίνακα με όνομα `randomRule` είναι `true` όπως φαίνεται στην γραμμή εντολής 10.

```
1. for (int d = 0; d < randomRule.length; d++) {  
2.  
3.     // generates random number from 0 to 1  
4.     random = rand.nextInt(2);  
5.  
6.     if (random == 0) {  
7.         randomRule[d] = false;  
8.         removedMetaRules.add(d);  
9.     } else {  
10.        randomRule[d] = true;  
11.    }  
12.  
13. }
```

Στις υπόλοιπες επαναλήψεις γίνεται τυχαία επιλογή της ενεργοποίησης και απενεργοποίησης μέχρι k κανόνων το οποίο αντιπροσωπεύετε από τη μεταβλητή `numberOfLoops`. Στην γραμμή εντολής 4 παρακάτω, παράγεται τυχαία ένας αριθμός από το 0 μέχρι τον αριθμό των μέτα-κανόνων, όπου αντιπροσωπεύει την αλλαγή της κατάστασης του συγκεκριμένου κανόνα. Αν είναι ενεργοποιημένος τότε η θέση του στον πίνακα `randomRule` από `true` γίνεται `false` και αφαιρείται από την λίστα απενεργοποιημένων κανόνων όπως φαίνεται από τις γραμμές εντολών 7-8. Διαφορετικά, ο κανόνας ενεργοποιείται και αφαιρείται από τη λίστα με τους απενεργοποιημένους κανόνες, όπως φαίνεται από τις γραμμές εντολών 10-11.

```

1. for (int i = 0; i < numberOfLoops; i++) {
2.
3.     // random here is the rule to be removed
4.     random = rand.nextInt(numrules);
5.
6.     if (randomRule[random]) {
7.         randomRule[random] = false;
8.         removedMetaRules.add(random);
9.     } else {
10.        randomRule[random] = true;
11.        removedMetaRules.remove(random);
12.    }
13.
14. }

```

Στον παρακάτω κώδικα φαίνεται ένα παράδειγμα υπολογισμού της εκπομπής διοξειδίου του άνθρακα. Το dt2 αντιπροσωπεύει την ημερομηνία και ώρα της καταγραφής από το σύνολο δεδομένων. Εάν η ώρα καταγραφής είναι μεταξύ του διαστήματος όπου ο μέτα-κανόνας ισχύει (ορίζεται από τη μεταβλητή ruleFrom και ruleTo) και εάν ο καιρός είναι ηλιόλουστος, τότε η συνολική εκπομπή διοξειδίου του άνθρακα αυξάνεται κατά span. Το span είναι η διάρκεια για την οποία τα φώτα ή το κλιματιστικό είναι ενεργοποιημένα. Η παρακάτω αύξηση της εκπομπής του διοξειδίου του άνθρακα γίνεται αν υπάρχει παραβίαση του μέτα-κανόνα διαφορετικά δεν αυξάνεται η εκπομπή διοξειδίου του άνθρακα.

```

1. if (dt2.getHours()>=ruleFrom && dt2.getHours()<=ruleTo) {
2.
3.     if (dataSplitted[4].equals("sky is clear")) {
4.         daily_ms_CO2 += span;
5.     }
6.
7. }

```

Κεφάλαιο 5

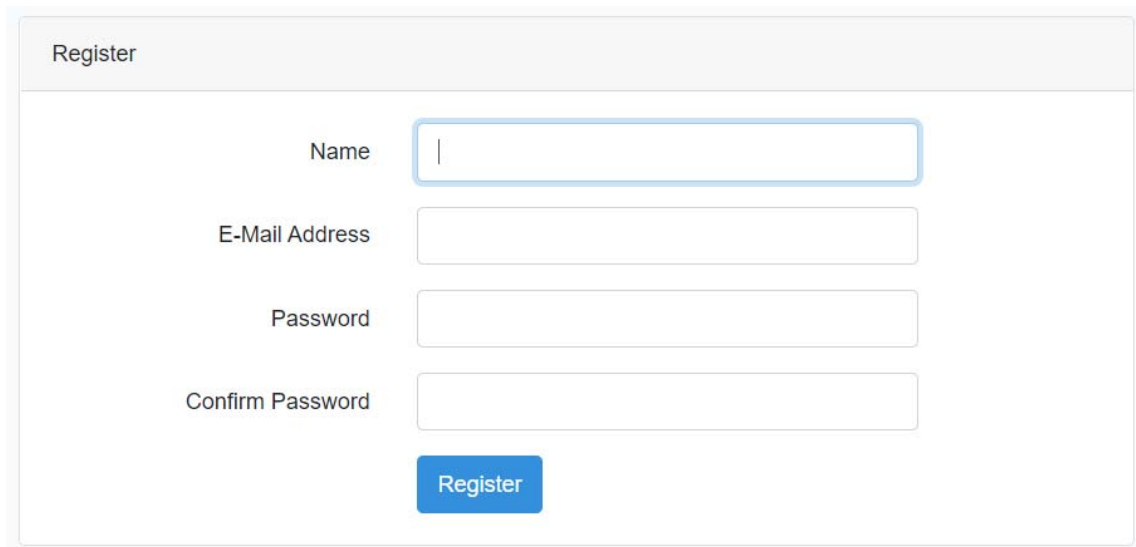
Διεπαφές Διαπροσωπείας

5.1 Δημιουργία λογαριασμού	40
5.2 Εισαγωγή του χρήστη στο σύστημα.....	41
5.3 Σελίδα Διαχείρισης Μέτα-Κανόνων.....	42
5.4 Εισαγωγή κανόνα Μέτα-Κανόνα.....	42
5.4 Επεξεργασία Μέτα-Κανόνα	43
5.6 Συσκευές συνδεδεμένες στο σύστημα	44
5.7 Αποτελέσματα εκτέλεσης	46

Σε αυτή την ενότητα περιγράφεται το σύστημα στο οποίο ένας χρήστης μπορεί να μπει αφού δημιουργήσει τον λογαριασμό του και εισάγει κάποιους κανόνες (Meta-rules) για τους οποίους χρησιμοποιεί ο αλγόριθμος Energy Planner (EP) και ο Green Planner (GP).

5.1 Δημιουργία λογαριασμού

Ο χρήστης για να δημιουργήσει λογαριασμό στο σύστημα με όνομα “Meta-rules Management System” πρέπει να εισάγει στο σύστημα το όνομα του, την ηλεκτρονική του διεύθυνση και τον κωδικό του.

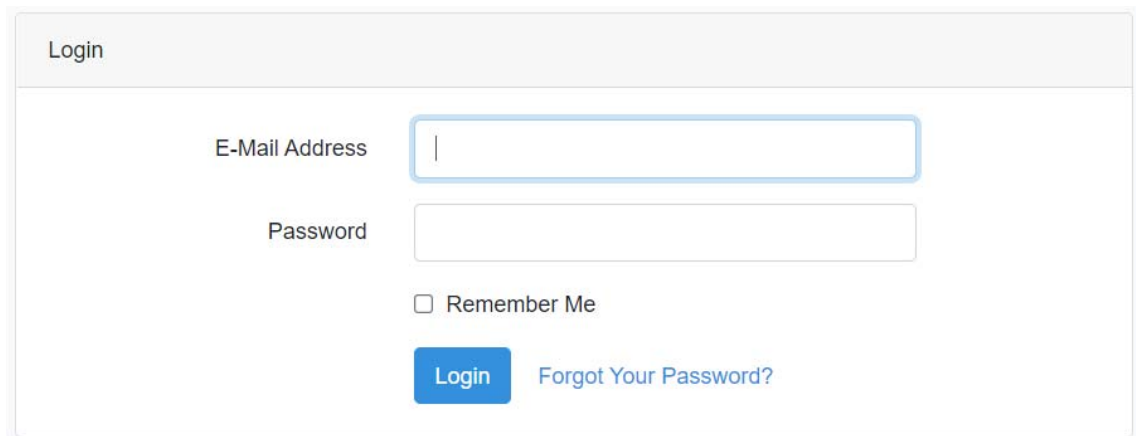


The image shows a 'Register' form with a light gray header bar containing the title 'Register'. Below the header, the form contains four input fields: 'Name', 'E-Mail Address', 'Password', and 'Confirm Password'. Each field is a white rectangle with a thin gray border. The 'Name' field is currently active, indicated by a blue border and a vertical cursor. Below the input fields is a blue button with the text 'Register' in white.

Σχήμα 5.1 Δημιουργία Λογαριασμού

5.2 Εισαγωγή του χρήστη στο σύστημα

Ο χρήστης για την εισαγωγή του στο σύστημα καλείται να γράψει την ηλεκτρονική του διεύθυνση και τον κωδικό του.



The image shows a 'Login' form with a light gray header bar containing the title 'Login'. Below the header, the form contains two input fields: 'E-Mail Address' and 'Password'. Each field is a white rectangle with a thin gray border. The 'E-Mail Address' field is currently active, indicated by a blue border and a vertical cursor. Below the input fields is a checkbox labeled 'Remember Me'. At the bottom, there is a blue button with the text 'Login' in white, followed by a link labeled 'Forgot Your Password?' in blue text.

Σχήμα 5.2 Εισαγωγή στο σύστημα

5.3 Σελίδα Διαχείρισης Μέτα-Κανόνων

Στη σελίδα διαχείρισης μέτα-κανόνων ο χρήστης μπορεί να προσθέσει, να αφαιρέσει και να επεξεργαστεί τους κανόνες προτιμήσεων τους.

Meta-rules configuration		
Meta-Rules Table		Create meta-rule
Description		
Morning Heat.	Edit	Delete
Morning Lights	Edit	Delete
Afternoon preheat house	Edit	Delete
Cosmetic Lights	Edit	Delete
Midday Lights	Edit	Delete
Spot lights	Edit	Delete
kWh Preferred Limit	Edit	Delete

Σχήμα 5.3 Κανόνες Meta-Rules

5.4 Εισαγωγή κανόνα Μέτα-Κανόνα

Ένας χρήστης μπορεί να εισάγει ένα καινούργιο μετα-κανόνα πατώντας το κουμπί “Create meta-rule” από την σελίδα “Meta-Rules”, βάζοντας μια περιγραφή του κανόνα,

το “action” το οποίο χωρίζεται σε τρεις κατηγορίες, ανάθεσης φωτός, ανάθεσης θερμοκρασίας, ανάθεσης ορίου kWh, ώρα για την οποία πρέπει να ισχύει ο κανόνας (στην περίπτωση ανάθεσης ορίου kWh δεν απαιτείται) και την τιμή.

Create meta-rule

Description

Action

Please select an action

Time Duration

From: --:--

To: --:--

Value

Submit

Σχήμα 5.4 Δημιουργία κανόνα Meta-Rule

5.4 Επεξεργασία Μέτα-Κανόνα

Σε περίπτωση λάθους, ή οποιοδήποτε λόγου ο χρήστης μπορεί να επεξεργαστεί ένα κανόνα πατώντας το κουμπί “Edit” από την σελίδα “Meta-Rules” και να τον αλλάξει. Πατώντας το κουμπί “Submit” μπορεί να αποθηκεύσει οποιαδήποτε αλλαγή έχει κάνει.

Edit meta-rule

Description

Morning Heat.

Action

Set temperature level

Time Duration

From: 04:00 To: 07:00

Value

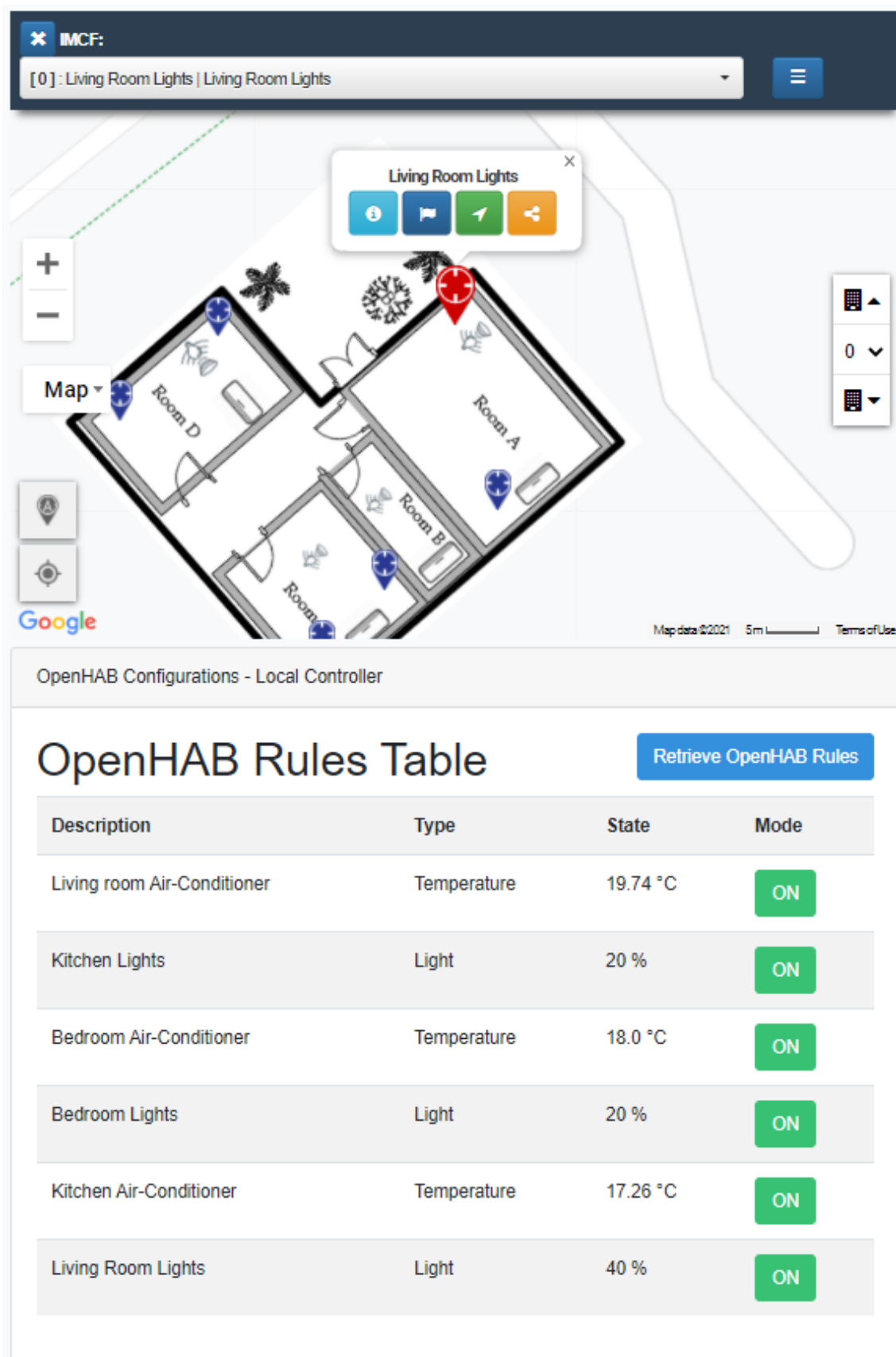
25

Submit

Σχήμα 5.5 Επεξεργασία κανόνα Meta-Rule

5.6 Συσκευές συνδεδεμένες στο σύστημα

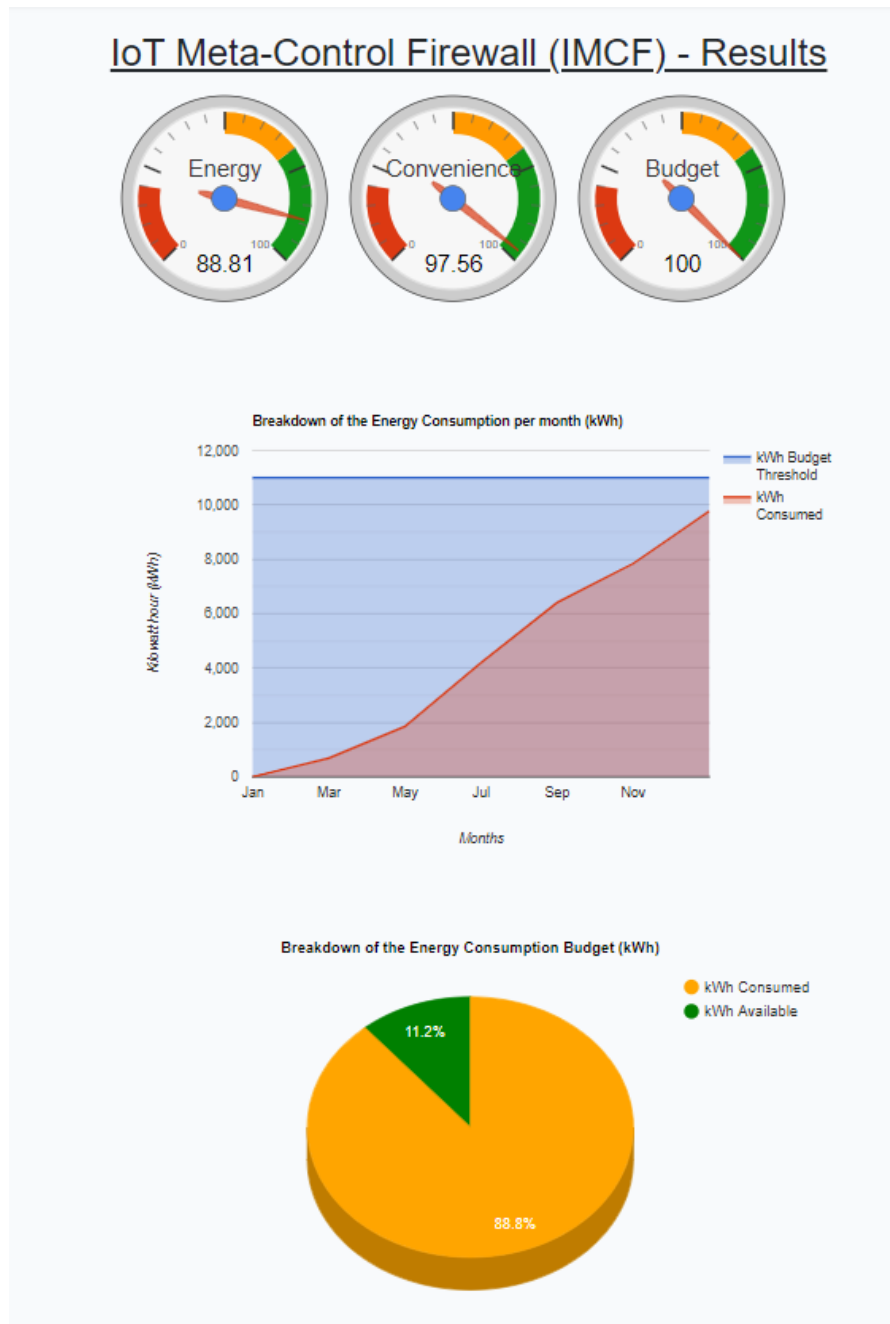
Ένας χρήστης μπορεί να δει τις διάφορες έξυπνες συσκευές που έχει συνδεδεμένες στο σύστημα IMCF μαζί με την κατάσταση τους (ON/OFF) στην σελίδα με όνομα “OpenHAB Local”. Οι διάφορες συσκευές προθέτονται από το OpenHAB και παρουσιάζονται μέσω της χρήσης του Anyplace.



Σχήμα 5.6 Έξυπνες συσκευές συνδεδεμένες στο σύστημα

5.7 Αποτελέσματα εκτέλεσης

Στην σελίδα με όνομα “Results” ο χρήστης μπορεί να δει τα αποτελέσματα που έχει παραγάγει το σύστημα. Στο σχήμα 5.7 μπορούμε να δούμε με την γραφική παράσταση την μηνιαία κατανάλωση ηλεκτρικού ρεύματος, το πόσο τοις εκατό είμαστε άνετα, το πόσο τοις εκατό βρισκόμαστε εντός προϋπολογισμού και πόσο τοις εκατό καταναλώσαμε ενέργεια.



Σχήμα 5.7 Αποτελέσματα Εκτέλεσης 1

IMCF Results	
Energy Planner - Results	
	Refresh Page
Description	Value
Kilowatt hour (kWh):	9768.74 kWh
Total Average Error:	2.44 %
Convenience Average Error:	2.44 %
Budget Average Error:	0.0 %
Standard Deviation:	3.20 %
Convenience Average Error for user: Soteris Constantinou	1.40 %
Convenience Average Error for user: Antonis Vasileiou	1.05 %

Σχήμα 5.8 Αποτελέσματα Εκτέλεσης 2

Στην περίπτωση όπου ο αλγόριθμος που χρησιμοποιήθηκε είναι ο Green Planner τα αποτελέσματα έχουν ως εξής (Τα αποτελέσματα είναι για σενάριο φοιτητικών εστιών).

IMCF Results	
Green Planner - Results	
	Refresh Page
Description	Value
Kilowatt hour (kWh):	384366.0 kWh
Kilowatt hour (kWh) with CO2 emissions:	333549.34 kWh
Kilowatt hour (kWh) without CO2 emissions:	50816.68 kWh
CO2 emissions:	149763.66 kgCO2e
Total Average Error:	0.93 %
Convenience Average Error:	0.93 %
Budget Average Error:	0.0 %
Standard Deviation:	1.05 %
Convenience Average Error for user: Antonis	0.93 %

Σχήμα 5.9 Αποτελέσματα Green Planner

Κεφάλαιο 6

Πειραματική Μεθοδολογία & Αποτίμηση

6.1 Πλατφόρμα	48
6.2 Σύνολα δεδομένων	49
6.3 Μετρικές	51
6.4 Αλγόριθμοι.....	51
6.4.1 Αλγόριθμοι NR και MR.....	51
6.4.2 Αλγόριθμος If-This-Then-That (IFTTT)	52
6.4.3 Αλγόριθμος Energy Planner	52
6.4.4 Αλγόριθμος Green Planner	52
6.5 Αξιολόγηση απόδοσης	53
6.6 Συμπεράσματα από την πειραματική εφαρμογή.....	55

Σε αυτή την ενότητα παρουσιάζεται η πειραματική μεθοδολογία που χρησιμοποιήθηκε και η αποτίμηση των αλγόριθμων. Αναφέρεται η πλατφόρμα που χρησιμοποιήθηκε την εκτέλεση των διάφορων πειραμάτων και τα διάφορα σύνολα δεδομένων, οι μετρικές που μετρήθηκαν για την πειραματική αποτίμηση καθώς και οι αλγόριθμοι που χρησιμοποιήθηκαν. Επίσης περιγράφεται η αξιολόγηση απόδοσης βάση πειράματος, που εκθέτει τα βασικά οφέλη του πλαισίου μας και του εσωτερικού αλγόριθμου GP και EP σε σύγκριση με τις βασικές τεχνικές.

6.1 Πλατφόρμα

Η αξιολόγησή πραγματοποιήθηκε στο κέντρο δεδομένων DMSL vCenter IaaS, ένα ιδιωτικό cloud, το οποίο περιλαμβάνει 8 IBM System x3550 M3, Lenovo ThinkSystem

SR590 Server και HP Proliant DL 360 G7 rackables με μονή υποδοχή (8 πυρήνες) ή διπλή υποδοχή (16 πυρήνες) Intel (R) Xeon (R) CPU E5620 @ 2.40GHz, αντίστοιχα. Αυτοί οι κεντρικοί υπολογιστές έχουν συλλογικά 300 GB κύριας μνήμης, 25 TB αποθήκευσης RAID-5 σε ένα IBM 3512 και συνδέονται μεταξύ τους μέσω ενός δικτύου Gigabit. Το κέντρο δεδομένων διαχειρίζεται μέσω ενός VMWare VSphere που συνδέεται με τους αντίστοιχους κεντρικούς υπολογιστές VMWare ESXi 6.7.0. Ο κόμβος υπολογιστών μας αποτελείται από μια εικόνα διακομιστή Ubuntu 18.04, που διαθέτει 8 GB RAM με 2 εικονικούς επεξεργαστές (@ 2.40GHz) Η εικόνα χρησιμοποιεί γρήγορους τοπικούς δίσκους RAID-5 LSI Logic SCSI 10K PM, μορφοποιημένους με VMFS 6 (μέγεθος μπλοκ 1MB).

6.2 Σύνολα δεδομένων

Υιοθετήθηκε μια πειραματική μεθοδολογία [11] που βασίζεται σε ίχνη στην οποία τα πραγματικά σύνολα δεδομένων τροφοδοτούνται στον προσομοιωτή μας που εκτελείται στο testbed. Αυτό επιτρέπει την επαναλαμβανόμενη εκτέλεση φόρτων εργασίας υπό διαφορετικές παραμέτρους ελέγχου. Πιο συγκεκριμένα, χρησιμοποιήθηκαν ανώνυμες μετρήσεις από ένα πραγματικό διαμέρισμα κατοικιών που αποτελείται από μια ποικιλία αισθητήρων, υπομέτρων και περίπου 5.668.878 αναγνώσεις (1,09 GB συνολικά). Πρόκειται για πραγματικά σύνολα δεδομένων οικιστικών δεδομένων που συλλέγονται από το "Κέντρο Προηγμένων Μελετών σε Προσαρμοσμένα Συστήματα" (CASAS) [22] στο Washington State University στη Σχολή Ηλεκτρολόγων Μηχανικών και Επιστήμης Υπολογιστών. Το CASAS εξυπηρετεί την κάλυψη ερευνητικών αναγκών σχετικά με τη δοκιμή των τεχνολογιών χρησιμοποιώντας πραγματικά δεδομένα μέσω της χρήσης ενός έξυπνου σπιτιού που βρίσκεται στην πανεπιστημιούπολη WSU Pullman. Το πραγματικό σύνολο δεδομένων πρόγνωσης καιρού αποκτήθηκε χρησιμοποιώντας το Weather API στον ιστότοπο του OpenWeatherMap και περιέχει ~ 5 χρόνια (2012-2017) δεδομένων υψηλής χρονικής ανάλυσης βάσει ωριαίων μετρήσεων διαφόρων χαρακτηριστικών καιρού, όπως θερμοκρασία, υγρασία, πίεση αέρα, περιγραφή καιρού, κατεύθυνση και ταχύτητα ανέμου.

Σύνολο δεδομένων θερμοκρασίας

Είναι ένα σύνολο δεδομένων μεγέθους 700 MB και περιέχει 3.555.238 αναγνώσεις σε δεύτερη βάση μεταξύ Οκτωβρίου 2013 και Δεκεμβρίου 2016. Οι μετρήσεις, οι οποίες καταγράφονται σε ένα οικιστικό διαμέρισμα ενός εθελοντή ενήλικα, περιλαμβάνουν μετρήσεις θερμοκρασίας και αισθητήρα πόρτας / παραθύρου.

Σύνολο δεδομένων φωτός

Είναι ένα σύνολο δεδομένων μεγέθους 416 MB και περιέχει 2.113.640 αναγνώσεις σε δεύτερη βάση μεταξύ Οκτωβρίου 2013 και Δεκεμβρίου 2016. Οι μετρήσεις, οι οποίες καταγράφονται σε οικιστικό διαμέρισμα ενός εθελοντή ενήλικα, περιλαμβάνουν μετρήσεις φωτός.

Σύνολο δεδομένων πρόγνωσης καιρού

Είναι ένα σύνολο δεδομένων μεγέθους 71,23 MB και περιέχει 271.561 αναγνώσεις ανά ώρα μεταξύ Οκτωβρίου 2012 και Νοεμβρίου 2017. Τα ιστορικά δεδομένα καιρού είναι διαθέσιμα για 30 πόλεις των ΗΠΑ και του Καναδά, καθώς και για 6 πόλεις του Ισραήλ, συμπεριλαμβανομένης της θερμοκρασίας, της υγρασίας, του αέρα πίεση, περιγραφή καιρού, κατεύθυνση και ταχύτητα ανέμου.

Σύνολο δεδομένων διαμερίσματος

Είναι ένα σύνολο δεδομένων για έναν χρήστη που κατοικεί σε διαμέρισμα το οποίο αποτελείται από ένα υπνοδωμάτιο, ένα μπάνιο και μια κουζίνα. Το διαμέρισμα είναι μεγέθους 50 m², διαθέτει μια ενιαία μονάδα για θέρμανση / ψύξη. Το σύνολο δεδομένων έχει μέγεθος 1,09 GB.

Σύνολο δεδομένων κατοικίας

Είναι ένα σύνολο δεδομένων κατοικίας που δημιουργήθηκε με την αναπαραγωγή, την ανάμειξη των μετρήσεων και τον πολλαπλασιασμό του πραγματικού συνόλου δεδομένων (σύνολο διαμερίσματος) επί 4 φορές. Η κατοικία είναι μεγέθους 200 m² και αποτελείται από 3 υπνοδωμάτια και 4 ενιαίες μονάδες για θέρμανση / ψύξη.

Σύνολο δεδομένων φοιτητικών εστιών

Είναι ένα σύνολο δεδομένων για μια φοιτητική εστία, το οποίο έχει δημιουργηθεί συνθετικά από τα σύνολα δεδομένων bootstrap. Αποτελείται από 50 κοιτώνες που αποτελούνται από δύο υπνοδωμάτια (10 m^2 / δωμάτιο) με κοινόχρηστο μπάνιο, κουζίνα και δύο χωριστές μονάδες. Το συνολικό μέγεθος των κοιτώνων είναι περίπου 2000m^2 και το σύνολο δεδομένων έχει μέγεθος 20 GB.

6.3 Μετρικές

Οι μετρικές [11] αποτελούνται από την κατανάλωση ενέργειας F_E , την εκπομπή διοξειδίου του άνθρακα F_T και το σφάλμα ευκολίας F_{CE} και ο χρόνος επεξεργαστή F_T όπου χρησιμοποιείται για σύγκριση στη μελέτη απόδοσης. Το F_T είναι ο χρόνος επεξεργασίας που απαιτείται από τον ελεγκτή για την εκτέλεση της λειτουργίας βελτιστοποίησης και τον υπολογισμό της εξόδου για όλους τους μέτα-κανόνες. Ο μέσος όρος και η τυπική απόκλιση των αποτελεσμάτων παρουσιάζονται με γραμμές σφάλματος σε όλες τις πειραματικές μελέτες που ακολουθούν, με βάση δέκα επαναλήψεις.

6.4 Αλγόριθμοι

Σε αυτή την υποενότητα αναφέρονται οι αλγόριθμοι που χρησιμοποιήθηκαν για την πειραματική αξιολόγηση.

6.4.1 Αλγόριθμοι NR και MR

Ο αλγόριθμος No-Rule (NR) [11] αγνοεί όλους τους μέτα-κανόνες του πίνακα μέτα-κανόνων και δεν τροποποιεί τη συμπεριφορά των αυτόνομων συσκευών. Από την άλλη ο αλγόριθμος Meta-Rule (MR) [11] αγνοεί την εξοικονόμηση κατανάλωσης ενέργειας και εκτελεί όλους τους μέτα-κανόνες με απληστία.

6.4.2 Αλγόριθμος If-This-Then-That (IFTTT)

Αυτή ο αλγόριθμος εκτελεί τους κανόνες προτιμήσεων τύπου IFTTT στο σύνολο δεδομένων διαμερίσματος και υπολογίζει το F_E και το F_{CE} .

IF	THIS	THEN	THAT
Season	Summer	Set Temper.	25
Season	Winter	Set Temper.	20
Weather	Sunny	Set Temper.	20
Weather	Cloudy	Set Light.	22
Weather	Sunny	Set Light.	0
Weather	Cloudy	Set Temper.	40
Temperature	>30	Set Temper.	23
Temperature	<10	Set Temper.	24
Light Level	>15	Set Light.	9
Door	Open	Set Light.	0

Πίνακας 6.1 Κανόνες IFTTT [11]

6.4.3 Αλγόριθμος Energy Planner

Για τον αλγόριθμο EP [11] έχει οριστεί αριθμός ενεργοποίησης / απενεργοποίησης κανόνων σε κάθε επανάληψη (k), ένα ποσοστό (τ) εξοικονόμησης και ο αριθμός των επαναλήψεων ($\tau_{\max}$).

6.4.4 Αλγόριθμος Green Planner

Για τον αλγόριθμο GP [21] έχει οριστεί αριθμός ενεργοποίησης / απενεργοποίησης κανόνων σε κάθε επανάληψη (k), ένα ποσοστό (τ) εξοικονόμησης και ο αριθμός των επαναλήψεων ($\tau_{\max}$).

6.5 Αξιολόγηση απόδοσης

Σε αυτήν την υποενότητα, αξιολογείτε η απόδοση του προτεινόμενου πλαισίου EP και GP έναντι όλων των αλγορίθμων σε όλα τα σύνολα δεδομένων που έχουν εισαχθεί, σε σχέση με την κατανάλωση ενέργειας και το σφάλμα ευκολίας. Το σχήμα 6.2 δείχνει την αντιστάθμιση μεταξύ της κατανάλωσης ενέργειας (F_E), του σφάλματος ευκολίας (F_{CE}) και του χρόνου εκτέλεσης της CPU (F_T) μεταξύ των βασικών προσεγγίσεων και της EP, ενώ το σχήμα 6.3 δείχνει την αντιστάθμιση μεταξύ της κατανάλωσης ενέργειας (F_E), του σφάλματος ευκολίας (F_{CE}), την εκπομπή CO_2 (F_T) και του χρόνου εκτέλεσης της CPU (F_T) μεταξύ των βασικών προσεγγίσεων και της GP. Η προσέγγιση NR απέκτησε το χειρότερο $F_{CE} = 62-72\%$ για όλα τα σύνολα δεδομένων, το καλύτερο $F_E = 0 \text{ kWh}$ και την καλύτερη εκπομπή CO_2 . Ο αλγόριθμος EP έλαβε ένα εντυπωσιακό F_{CE} περίπου 2-4% και το τρίτο χαμηλότερο F_E . Ο αλγόριθμος GP έλαβε ένα F_{CE} περίπου 6.5-8%, το δεύτερο χαμηλότερο F_E και τη δεύτερη μικρότερη εκπομπή CO_2 μετά την NR. Οι αλγόριθμοι IFTTT & MR είναι άπληστοι όσον αφορά την κατανάλωση ενέργειας, επομένως η κατανάλωση kWh και η εκπομπή CO_2 τους είναι πολύ υψηλή. Η κύρια διαφορά μεταξύ των δύο είναι ότι το IFTTT έχει $F_{CE} = 26\%$ στο διαμέρισμα, $F_{CE} = 29\%$ στην περίπτωση της κατοικίας και $F_{CE} = 39\%$ στην περίπτωση των φοιτητικών κοιτώνων, ενώ η MR ικανοποιεί όλους τους μέτα-κανόνες, έτσι το F_{CE} του είναι 0%, το οποίο είναι το καλύτερο δυνατό.

Ο αλγόριθμος EP για την περίπτωση του διαμερίσματος κατάφερε να εξοικονομήσει έως και 10% της ενέργειας, που είναι περίπου 9500 kWh για τον προτιμώμενο ενεργειακό προϋπολογισμό των 11000 kWh και για τα τρία χρόνια, με F_{CE} που έφτασε σε λογικές τιμές των 2-3%. Στην περίπτωση ενός σπιτιού, ο προτιμώμενος ενεργειακός προϋπολογισμός διαμορφώθηκε στα 25500 kWh και για τα τρία χρόνια και η EP κατάφερε να επιτύχει περίπου 22300 kWh, με F_{CE} περίπου 2-2,5%. Στην περίπτωση των κοιτώνων, ο προτιμώμενος προϋπολογισμός ενέργειας διαμορφώθηκε στα 480.000 kWh και για τα τρία χρόνια και η EP κατάφερε να επιτύχει περίπου 410.000 kWh, με την τιμή του F_{CE} να φτάνει την λογική τιμή των 2,5-3%. Εδώ είναι σημαντικό να παρατηρηθεί, ότι η διαφορά μεταξύ του MR και του EP όσον αφορά την κατανάλωση ενέργειας, είναι σχετικά υψηλή και συγκεκριμένα $\approx 5.000 \text{ kWh}$ για το σύνολο δεδομένων του

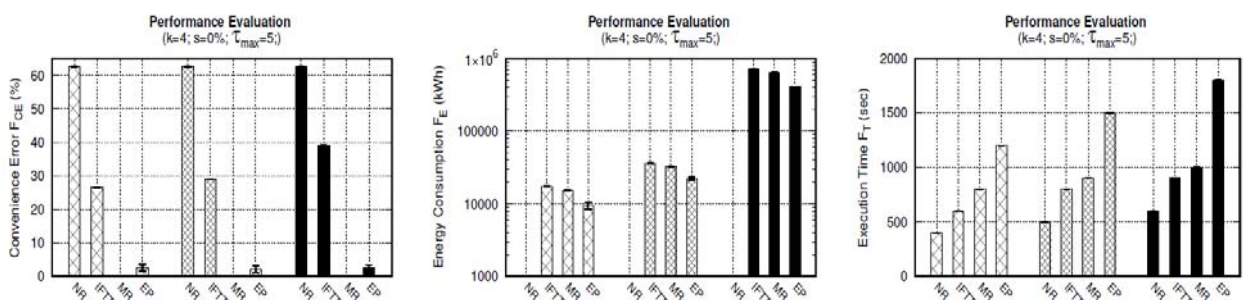
διαμερίσματος, $\approx 10.000 \text{ kWh}$ για το σύνολο δεδομένων κατοικίας και $\approx 150.000 \text{ kWh}$ για το σύνολο δεδομένων των φοιτητικών κοιτώνων.

Ο αλγόριθμος GP για την περίπτωση του διαμερίσματος κατάφερε να εξοικονομήσει έως και 10% της ενέργειας, που είναι περίπου 9500 kWh για τον προτιμώμενο ενεργειακό προϋπολογισμό των 11000 kWh και για τα τρία χρόνια, με F_{CE} που έφτασε σε λογικές τιμές των 6.5-7% και εκπομπές $\text{CO}_2 \approx 3600 \text{ Kg}$. Στην περίπτωση ενός σπιτιού, ο προτιμώμενος ενεργειακός προϋπολογισμός διαμορφώθηκε στα 25500 kWh και για τα τρία χρόνια και η GP κατάφερε να επιτύχει περίπου 22300 kWh, με F_{CE} περίπου 7-7,5% και εκπομπές $\text{CO}_2 \approx 8700 \text{ Kg}$. Στην περίπτωση των κοιτώνων, ο προτιμώμενος προϋπολογισμός ενέργειας διαμορφώθηκε στα 480.000 kWh και για τα τρία χρόνια και η GP κατάφερε να επιτύχει περίπου 410.000 kWh, με την τιμή του F_{CE} να φτάνει την λογική τιμή των 7,5-8% και εκπομπές $\text{CO}_2 \approx 135,000 \text{ Kg}$. Το κόστος ανά kWh στην Κύπρο είναι 0.181 ευρώ [23], έτσι ο πίνακας που ακολουθεί μας δείχνει την εξοικονόμηση των χρημάτων σε σχέση με το όριο που έχει οριστεί.

Τύπος	Ενεργειακό Όριο	Κατανάλωση με EP/GP	Εξοικονόμηση Χρημάτων
Διαμέρισμα	11000 kWh	9500 kWh	271.5 ευρώ
Κατοικία	25500 kWh	22300 kWh	579.2 ευρώ
Φοιτητικές Εστίες	480000 kWh	410000 kWh	12,670 ευρώ

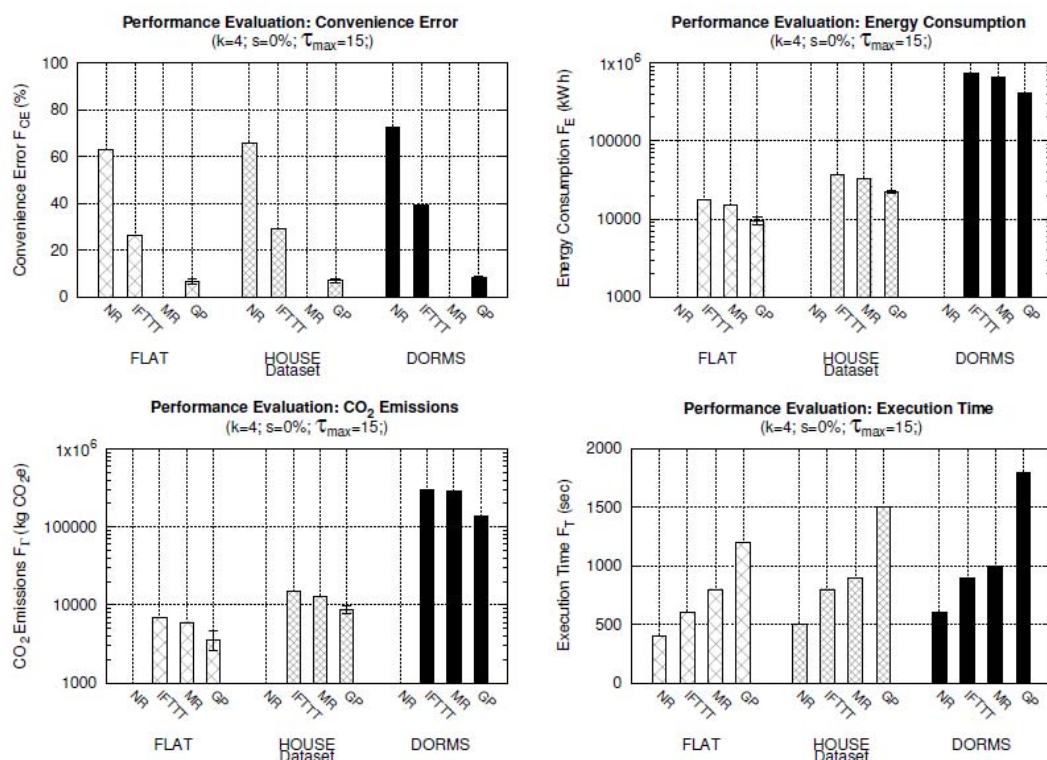
Πίνακας 6.2 Εξοικονόμηση Χρημάτων με χρήση των αλγόριθμων

Ο ταχύτερος χρόνος εκτέλεσης επιτεύχθηκε από την NR, καθώς απλώς εκτελεί έναν υπολογισμό σφάλματος αγνοώντας όλους τους κανόνες. Η προσέγγιση αναρρίχησης του



Σχήμα 6.1 Αξιολόγηση απόδοσης Energy Planner [4]

λόφου της EP, από την άλλη, αναζητά το χώρο λήψης αποφάσεων για μια λύση που θα βελτιστοποιήσει την άνεση του χρήστη και θα ικανοποιήσει τον ενεργειακό περιορισμό, ταυτόχρονα, που είναι μια πολύ πιο χρονοβόρα διαδικασία. Ο αλγόριθμος GP χρησιμοποιεί την προσομοιωμένη προσέγγιση ανόπτησης αναζητώντας μια βέλτιστη λύση για τον χρήστη λαμβάνοντας υπόψη τον προτιμώμενο περιορισμό του επιτρεπόμενου ενεργειακού προϋπολογισμού και τον περιορισμό των εκπομπών CO₂, επομένως είναι η πιο χρονοβόρα μέθοδος. Τέλος, η άπληστη προσέγγιση του MR εστιάζει μόνο στην ελαχιστοποίηση του σφάλματος ευκολίας, που σημαίνει την εκτέλεση όλων των μέτα-κανόνων χωρίς επαναλαμβανόμενες διαδικασίες ή υπολογισμούς, αφού $F_{CE} = 0\%$.



Σχήμα 6.2 Αξιολόγηση απόδοσης Green Planner [21]

6.6 Συμπεράσματα από την πειραματική εφαρμογή

Οι αλγόριθμοι έχουν καταφέρει να μείνουν εντός του ενεργειακού στόχου που έθεσαν οι χρήστες και να γίνει εξοικονόμηση 10% της ενέργειας κρατώντας τους χρήστες κατά 2-8% μη ευχαριστημένους. Το σφάλμα ευκολίας F_{CE} στους αλγόριθμους Energy Planner

και Green Planner φτάνει στα 2-3% και 6.5%-8% αντίστοιχα. Αυτό σημαίνει ότι, οι χρήστες μένουν αρκετά ικανοποιημένοι χωρίς να επηρεάσει πολύ την ευκολία στην καθημερινότητα τους, δηλαδή ότι διατηρούνται οι μέτα-κανόνες που έθεσαν ενώ το ενεργειακό όριο που έθεσαν δεν ξεπεράστηκε. Έτσι καταλήγουμε στο συμπέρασμα πως οι αλγόριθμοι έχουν εκπληρώσει με επιτυχία τον στόχο που θέσαμε, το να μειώσουμε την κατανάλωση ενέργειας και τις εκπομπές ρίπων θερμοκηπίου.

Κεφάλαιο 7

Συμπεράσματα

7.1 Συμπεράσματα	57
7.2 Μελλοντική δουλειά	58

7.1 Συμπεράσματα

Ανακεφαλαιώνοντας, αυτή η διπλωματική εργασία είχε σκοπό την επίλυση του προβλήματος της επίτευξης μακροχρόνιων στόχων ενέργειας σε ένα έξυπνο κτήριο, ούτως ώστε να συνεισφέρουμε θετικά στις διάφορες προσπάθειες των χωρών να μειώσουν την κατανάλωση ενέργειας και τη μείωση ρίπων διοξειδίου του άνθρακα, για τη μείωση του φαινομένου του θερμοκηπίου. Για την επίλυση αυτού του προβλήματος υλοποιήθηκε το πλαίσιο κανόνων προτιμήσεων για επίτευξη μακροχρόνιων στόχων όπως η συνολική κατανάλωση ενέργειας σε ένα έξυπνο κτήριο.

Η υλοποίηση αυτού του πλαισίου συμπεριλαμβάνει την υλοποίηση των αλγορίθμων του συστήματος μέτα-κανόνων διαδικτύου των πραγμάτων (IMCF). Αρχικά μελετήθηκαν οι διάφοροι αλγόριθμοι τεχνητής νοημοσύνης που χρησιμοποιούνται στους αλγόριθμους Energy Planner και Green Planner ούτως ώστε να επιτευχθεί η καλύτερη κατανόηση τους, καθώς και κάποιες έξυπνες συσκευές για τις οποίες μπορεί να εφαρμοστεί το πλαίσιο μας. Επίσης όπως έχουν μελετηθεί οι ροές αυτοματισμού (RAW) και το if-this-then-that που χρησιμοποιήθηκαν για την υλοποίηση των διάφορων αλγορίθμων. Οι αλγόριθμοι κτίστηκαν βάση της αρχιτεκτονικής και ενσωματώθηκαν στο ολοκληρωμένο σύστημα IMCF.

Βάση των διάφορων πειραμάτων και αποτιμήσεων των αλγορίθμων που έγιναν, καταλήξαμε στο συμπέρασμα ότι το πλαίσιο το οποίο αναπτύχθηκε είναι ένας καλός τρόπος για επίτευξη των μακροχρόνιων στόχων βάση κανόνων που μπορεί να θέσουν οι χρήστες, αφού καταφέραμε να επιτύχουμε 10% μείωση στην κατανάλωση ενέργειας κρατώντας τους χρήστες αρκετά ικανοποιημένους καθώς το σφάλμα ευκολίας έφτασε μόνο στο 2-4% στην περίπτωση χρήσης του αλγορίθμου Energy Planner και στο 7-8% στην περίπτωση χρήσης του αλγορίθμου Green Planner. Αποτέλεσμα αυτού είναι η επίτευξη του στόχου μείωσης κατανάλωσης ενέργειας και μείωσης της εκπομπής ρίπων θερμοκηπίου.

7.2 Μελλοντική δουλειά

Το πλαίσιο εκτέλεσης κανόνων προτιμήσεων έχει αποδειχτεί ως ένας αποδοτικός τρόπος για μείωση της κατανάλωσης ενέργειας και μείωσης των ρίπων του διοξειδίου του άνθρακα. Ως εκ τούτου, στο μέλλον το πλαίσιο αυτό μπορεί να εφαρμοστεί σε ένα πραγματικό έξυπνο κτήριο μέσω ενσωμάτωσης σε έξυπνες πλατφόρμες ενεργοποίησης και να μελετηθούν τα πραγματικά οφέλη του. Επίσης στο πλαίσιο, μπορεί να αναπτυχθεί το CMC controller [11] το οποίο μπορεί να ελέγχει το πλαίσιο μέσω ενός cloud, για να μπορούν οι χρήστες να το χρησιμοποιούν εκτός του προσωπικού δικτύου τους. Ακόμα στο μέλλον μπορεί να μελετηθούν μέθοδοι αναγνώρισης φόρτου εργασίας για συσκευές που καταναλώνουν ενέργεια και πώς να αναπρογραμματιστούν με φιλικό προς το περιβάλλον τρόπο.

Αναφορές

- [1] «Paris Agreement,» United Nations Climate Change, 2015. [Ηλεκτρονικό]. Available: <https://unfccc.int/process-and-meetings/the-paris-agreement/the-paris-agreement>.
- [2] «Green Deal,» European Commission, 2020. [Ηλεκτρονικό]. Available: https://ec.europa.eu/info/strategy/priorities-2019-2024/european-green-deal_en.
- [3] «Saves,» Erasmus Student Network, 2014. [Ηλεκτρονικό]. Available: <https://esn.org/saves>.
- [4] “Sunny home manager 2.0,” SMA, 2019. [Online]. Available: <https://www.sma.de/en/products/monitoring-control/sunny-home-manager-20.html>.
- [5] «SMA Developer,» SMA, [Ηλεκτρονικό]. Available: <https://www.sma.de/en/products/sma-developer.html>.
- [6] «Google Nest Thermostat,» Google, [Ηλεκτρονικό]. Available: https://store.google.com/us/product/nest_learning_thermostat_3rd_gen?hl=en-US.
- [7] «Kasa Smart WiFi Plug Slim with Energy Monitoring,» Tp-Link, [Ηλεκτρονικό]. Available: <https://www.kasasmart.com/us/products/smart-plugs/kasa-smart-plug-slim-energy-monitoring-kp115>.
- [8] Wikipedia, «EEBUS,» [Ηλεκτρονικό]. Available: <https://en.wikipedia.org/wiki/EEBUS>.
- [9] «Technology - EEBus Initiative e.V.,» EEBus, [Ηλεκτρονικό]. Available: <https://www.eebus.org/technology/>.
- [10] «Row Automation Workflow,» [Ηλεκτρονικό]. Available: <https://intercom.help/mylogiwa/en/articles/4619442-what-is-a-workflow-automation-rule>.
- [11] Soteris Constantinou, Andreas Konstantinidis, Demetrios Zeinalipour-Yazti and Panos K. Chrysanthis, «The IoT Meta-Control Firewall,» σε 37th IEEE International Conference on Data Engineering, Chania, Crete, pp.1-12, 2021.
- [12] S. Heo, S. Song, J. Kim, and H. Kim., «Rt-ifttt: Real-time iot framework,» σε *2017 IEEE Real-Time Systems Symposium (RTSS)*, Paris, 2017.
- [13] «What is a Smart Actuator,» [Ηλεκτρονικό]. Available: <https://www.infobloom.com/what-is-a-smart-actuator.htm>.

- [14] «What is smart actuation?,» [Ηλεκτρονικό]. Available:
https://www.thomsonlinear.com/micro/smart-actuation_eng/what-is-smart-actuation.html.
- [15] «Energy demand management,» [Ηλεκτρονικό]. Available:
https://en.wikipedia.org/wiki/Energy_demand_management#Household_scale.
- [16] «Local Search,» Wikipedia, [Ηλεκτρονικό]. Available:
[https://en.wikipedia.org/wiki/Local_search_\(optimization\)](https://en.wikipedia.org/wiki/Local_search_(optimization)).
- [17] «Hill Climbing,» Wikipedia, [Ηλεκτρονικό]. Available:
https://en.wikipedia.org/wiki/Hill_climbing.
- [18] «Travelling Salesman,» Wikipedia, [Ηλεκτρονικό]. Available:
https://en.wikipedia.org/wiki/Travelling_salesman_problem.
- [19] «Hill climbing algorithm in AI,» javatpoint, [Ηλεκτρονικό]. Available:
<https://www.javatpoint.com/hill-climbing-algorithm-in-ai>.
- [20] «Simulated Annealing,» Wikipedia, [Ηλεκτρονικό]. Available:
https://en.wikipedia.org/wiki/Simulated_annealing.
- [21] Soteris Constantinou, Andreas Konstantinidis, Demetrios Zeinalipour-Yazti and Panos K. Chrysanthis, «Green Planning of IoT Rule Automation Workflows,» 2021.
- [22] A. Brown, «Interactions,» σε ACM interactions, vol 9, no 2, pp 51-53, 2002.
- [23] «Cyprus electricity prices,» GlobalPetrolPrices.com, [Ηλεκτρονικό]. Available:
https://www.globalpetrolprices.com/Cyprus/electricity_prices/.

Παράρτημα Α

Στο παράρτημα αυτό παραθέτετε ο ολοκληρωμένος κώδικας των αλγορίθμων του πλαισίου εκτέλεσης κανόνων, που υλοποιήθηκαν στα πλαίσια της παρούσας διπλωματικής εργασίας.

importData.java

```
1  package iotprojectpackage;
2
3  import java.io.BufferedReader;
4  import java.io.File;
5  import java.io.FileWriter;
6  import java.io.InputStreamReader;
7  import java.net.*;
8  import java.util.ArrayList;
9  import java.util.Scanner;
10
11 public class importData {
12
13     private ArrayList<String[]> IotData = new ArrayList<String[]>();
14     private ArrayList<String[]> IFTTrules = new ArrayList<String[]>();
15     private ArrayList<String[]> MRTrules = new ArrayList<String[]>();
16     private String pathIFTTrules;
17     private String pathMRTrules;
18     private String pathIoTdata;
19
20     public importData(String mrtpath, String IFTTpath, String pathData) {
21         pathIFTTrules = IFTTpath;
22         pathMRTrules = mrtpath;
23         pathIoTdata = pathData;
24     }
25
26     public void get_updated_MRTrules() {
27
28         String urlString = "http://10.16.30.73/api/get-metarules";
29
30         try {
31
32             StringBuilder result = new StringBuilder();
33             URL url = new URL(urlString);
34             HttpURLConnection conn = (HttpURLConnection) url.openConnection();
35             conn.setRequestMethod("GET");
36             conn.connect();
37             int responsecode = conn.getResponseCode();
38             if (responsecode != 200) {
39                 throw new RuntimeException("HttpStatusCode: " + responsecode);
40             } else {
41                 BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
42                 String line;
43                 while ((line = br.readLine()) != null) {
44                     result.append(line);
45                 }
46
47                 br.close();
48
49             }
50         }
51     }
52 }
```

```

49         result.deleteCharAt(result.length() - 1);
50         result.deleteCharAt(0);
51         String[] stemp = result.toString().split(",");
52         StringBuilder stringbld = new StringBuilder();
53
54         for (String s : stemp) {
55             s = s.replace("\"", "");
56             s = s.replace("\n", "");
57             String stemp2[] = s.split(",");
58             for (String s1 : stemp2) {
59                 // System.out.println(s1);
60                 String stemp3[] = s1.split(":");
61                 if (stemp3[0].equals("time")) {
62                     if (stemp3[1].equals("kWh")) {
63                         stringbld.append("kWh");
64                         stringbld.append("|");
65                     } else {
66                         stringbld.append(stemp3[1]);
67                         stringbld.append(":");
68                         stringbld.append(stemp3[2]);
69                         stringbld.append(":");
70                         stringbld.append(stemp3[3]);
71                         stringbld.append("|");
72                     }
73                 } else if (stemp3[0].equals("action")) {
74                     stringbld.append(stemp3[1]);
75                     stringbld.append("|");
76                 } else if (stemp3[0].equals("value")) {
77                     stringbld.append(stemp3[1]);
78                     stringbld.append("|");
79                     stringbld.append(stemp3[1]);
80                     stringbld.append("\n");
81                 }
82             }
83             // System.out.println(s);
84         }
85         // System.out.println(stringbld);
86         FileWriter myWriter = new FileWriter(pathMRTrules);
87         myWriter.write("TIME| TEMPER./LIGHT | FROM | TO | \n");
88         myWriter.write(
89             "-----\n"
90         );
91
92         myWriter.write(stringbld.toString());
93         myWriter.close();
94     }
95 }
96

```

```

97     } catch (Exception e) {
98         e.printStackTrace();
99         System.out.println(e.getMessage());
100     }
101 }
102
103
104 public void get_updated_MRTrules_GP() {
105     // call api for data for location and date of today
106     // put in dame structure as data I already have
107     // put date and T105 and the value of temp the other put whatever
108     // First store in text file and then import them like before
109     // String API_KEY = "c5c61330eca67b966249ef80724111a6";
110     // String LOCATION = "Limassol,CY";
111     String urlString = "http://10.16.30.73/imcftp/api/get-metarules";
112
113     try {
114
115         StringBuilder result = new StringBuilder();
116         URL url = new URL(urlString);
117         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
118         conn.setRequestMethod("GET");
119         conn.connect();
120         int responsecode = conn.getResponseCode();
121         if (responsecode != 200) {
122             throw new RuntimeException("HttpStatusCode: " + responsecode);
123         } else {
124             BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
125             String line;
126             while ((line = br.readLine()) != null) {
127                 result.append(line);
128             }
129
130             br.close();
131
132             result.deleteCharAt(result.length() - 1);
133             result.deleteCharAt(0);
134             String[] stemp = result.toString().split(",");
135             StringBuilder stringbld = new StringBuilder();
136
137             for (String s : stemp) {
138                 s = s.replace("\"", "");
139                 s = s.replace("\n", "");
140                 String stemp2[] = s.split(",");
141                 for (String s1 : stemp2) {
142                     // System.out.println(s1);
143                     String stemp3[] = s1.split(":");
144                     if (stemp3[0].equals("time")) {

```

```

145         if (stemp3[1].equals("kWh")) {
146             stringbld.append("kWh");
147             stringbld.append("|");
148         } else {
149             stringbld.append(stemp3[1]);
150             stringbld.append(":");
151             stringbld.append(stemp3[2]);
152             stringbld.append(":");
153             stringbld.append(stemp3[3]);
154             stringbld.append("|");
155         }
156     } else if (stemp3[0].equals("action")) {
157         stringbld.append(stemp3[1]);
158         stringbld.append("|");
159     } else if (stemp3[0].equals("value")) {
160         stringbld.append(stemp3[1]);
161         stringbld.append("|");
162         stringbld.append(stemp3[1]);
163         stringbld.append("\n");
164     }
165 }
166 // System.out.println(s);
167
168 }
169 // System.out.println(stringbld);
170 FileWriter myWriter = new FileWriter(pathMRTrules);
171 myWriter.write("TIME| TEMPER./LIGHT | FROM | TO | \n");
172 myWriter.write(
173     "-----
174
175     myWriter.write(stringbld.toString());
176     myWriter.close();
177
178 }
179
180 } catch (Exception e) {
181     e.printStackTrace();
182     System.out.println(e.getMessage());
183 }
184
185 }
186
187 public void import_IoT_data() {
188
189     int counter = 0;
190     String line;
191
192     // Read the file and display it line by line.
193
194
195     File file = new File(pathIoTdata);
196
197     try {
198         System.out.println("Importing Data:\t" + file.getName());
199         long starttime = System.nanoTime();
200         Scanner inputFile = new Scanner(file);
201         // loop through every line
202
203         while (inputFile.hasNextLine()) {
204             line = inputFile.nextLine();
205             // String[] col = line.split("|");
206
207             String temp[] = new String[7];
208             if (counter > 1) {
209                 String[] col = line.split("\\|");
210                 temp[0] = col[0].trim();
211                 temp[1] = col[1].trim();
212                 temp[2] = col[2].trim();
213                 temp[3] = col[3].trim();
214                 temp[4] = col[4].trim();
215                 temp[5] = col[5].trim();
216                 temp[6] = col[6].trim();
217
218                 IotData.add(temp); // fills a list with all the input IoT data
219             }
220
221             counter++;
222         } // end of while loop
223
224         inputFile.close();
225
226         long finishtime = System.nanoTime() - starttime;
227         System.out.println("Import Completed!");
228         System.out.println("Import Dataset Time:\t" + finishtime / 1000000 + "ms");
229     } catch (Exception e) {
230         System.out.println("Error with Import of IoT data.");
231         System.out.println(e);
232     }
233
234     System.out.println("Number of records imported: " + IotData.size());
235
236 }
237
238 public void import_IFTTT_rules() {
239     int counter = 0;
240     String line;
241
242     // Read the file and display it line by line.

```

```

241 File file = new File(pathIFTTrules);
242
243 System.out.println("Importing IFTT Rules:\t" + file.getName());
244 try {
245     long starttime = System.nanoTime();
246     Scanner inputFile = new Scanner(file);
247
248     // loop through every line
249     while (inputFile.hasNextLine()) {
250         line = inputFile.nextLine();
251         String[] col = line.split("\\|");
252
253         String temp[] = new String[8];
254         if (counter > 1) {
255             temp[0] = col[0].trim();
256             temp[1] = col[1].trim();
257             temp[2] = col[5].trim();
258             temp[3] = col[3].trim();
259             temp[4] = col[6].trim();
260             temp[5] = col[7].trim();
261             temp[6] = col[8].trim();
262             temp[7] = col[2].trim();
263             IFTTrules.add(temp);
264         } // end of inner while loop
265         counter++;
266     } // end of outer while loop
267     long finishtime = System.nanoTime() - starttime;
268     System.out.println("Import Completed!");
269     System.out.println("Import IFTT rules Time:\t" + finishtime / 1000000 + "ms");
270     inputFile.close();
271     System.out.printf("There were %d lines.\n", counter);
272 } catch (Exception e) {
273     System.out.println("Error with Import of IFTT rules.");
274     System.out.println(e);
275 }
276 // return IFTTrules;
277 }
278
279 public void import_MRT_rules() {
280     int counter = 0;
281     String line;
282
283     // Read the file and display it line by line.
284
285     File file = new File(pathMRTrules);
286     System.out.println("Importing MRT Rules:\t" + file.getName());
287     try {
288         long starttime = System.nanoTime();
289
290         Scanner inputFile = new Scanner(file);
291
292         // loop through every line
293         while (inputFile.hasNextLine()) {
294             line = inputFile.nextLine();
295             String[] col = line.split("\\|");
296
297             String temp[] = new String[4];
298             if (counter > 1) {
299                 temp[0] = col[0].trim();
300                 temp[1] = col[2].trim();
301                 temp[2] = col[3].trim();
302                 temp[3] = col[1].trim();
303                 MRTrules.add(temp);
304             } // end of if
305             counter++;
306         } // end of while loop
307
308         long finishtime = System.nanoTime() - starttime;
309         System.out.println("Import Completed!");
310         System.out.println("Import MRT rules Time:\t" + finishtime / 1000000 + "ms");
311         inputFile.close();
312         System.out.printf("There were %d lines.\n", counter);
313     } catch (Exception e) {
314         System.out.println("Error with Import of MRT rules.");
315         System.out.println(e);
316     }
317
318 } // END OF - Meta-rule Config SCENARIO 2
319
320 public ArrayList<String[]> getIotData() {
321     return IotData;
322 }
323
324 public ArrayList<String[]> getIFTTrules() {
325     return IFTTrules;
326 }
327
328 public ArrayList<String[]> getMRTrules() {
329     return MRTrules;
330 }
331
332 }
333
334
335
336

```

noIFTT.java

```

1  package iotprojectpackage;
2
3  import java.io.FileWriter;
4  import java.math.RoundingMode;
5  import java.text.DecimalFormat;
6  import java.util.ArrayList;
7
8  public class noIFTT {
9
10     private static ArrayList<String[]> IotData = new ArrayList<String[]>();
11     private ArrayList<String[]> IotData3 = new ArrayList<String[]>(); // for use in noIFTT function
12     private ArrayList<String> TempIotData = new ArrayList<String>();
13     private static ArrayList<String[]> IFTRules;
14     private static ArrayList<String[]> MRTRules;
15     private String errorLine = "";
16     private double formaliseMaxError = 0;
17     private String pathNoIFTTresults;
18
19     public noIFTT(ArrayList<String[]> iftrules, ArrayList<String[]> mrtrules, ArrayList<String[]> data,
20         String pathForResults) {
21         IFTRules = new ArrayList<String[]>(iftrules);
22         MRTRules = new ArrayList<String[]>(mrtrules);
23         IotData = new ArrayList<String[]>(data);
24         pathNoIFTTresults = pathForResults;
25     }
26
27     public void printLine(String[] array) {
28         int length = array.length;
29         int count = 0;
30
31         for (String temp : array) {
32             if (count == length - 1) {
33                 System.out.println(temp);
34             } else {
35                 System.out.print(temp + "|");
36             }
37             count++;
38         }
39     }
40
41     public String makeDataLine(String[] array) {
42         String s = "";
43         String separator = "|";
44         int count = 0;
45         int length = array.length;
46
47         for (String temp : array) {
48             if (count == length - 1) {
49                 s = s.concat(temp);
50             } else {
51                 s = s.concat(temp);
52                 s = s.concat(separator);
53             }
54             count++;
55         }
56
57         return s;
58     }
59
60     // returns formaliseMaxError value
61     public double method() {
62
63         String listRanges[][] = MRTRules.toArray(new String[0][]);
64
65         double dTotalError = 0;
66         double dTempError = 0;
67         double dLineError = 0;
68
69         long starttime = System.nanoTime();
70         double maxError = 0;
71         double minError = 100;
72         boolean errorFound = false;
73         int errorCounter = 0;
74
75         int line = 0;
76         try {
77
78             for (int i = 0; i < IotData.size(); i++) {
79                 // gets the total record line error in regards of all the ranges
80                 dLineError = 0;
81
82                 for (int k = 0; k < listRanges.length; k++) {
83                     String[] rangesSplitted = listRanges[k];
84                     errorLine = makeDataLine(IotData.get(i));
85
86                     String[] IotDataSplitted2 = IotData.get(i);
87                     // gets T for temperatures
88                     if (IotDataSplitted2[2].startsWith("T")) {
89                         // checks time and temperature and if season is empty
90
91

```

```

91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136

```

```

    if ((!(rangesSplitted[0].equals("SUMMER")) || !(rangesSplitted[0].equals("WINTER"))
        && (rangesSplitted[3].equals("TEMPER.")))) {

        if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
            .parseDouble(rangesSplitted[0].substring(0, 2))
            && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
            .parseDouble(rangesSplitted[0].substring(6, 8)))
            && Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
            && Double.parseDouble(IotDataSplitted2[3]) <= Double
            .parseDouble(rangesSplitted[2])) {
            IotData3.add(IotData.get(i));
            // its already on the list no need to add it twice
            break;
        } else {
            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
            if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
                dTotalError += Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                dLineError += Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                if (dLineError > maxError) {
                    maxError = dLineError;
                }
                if (dLineError < minError) {
                    minError = dLineError;
                }
                errorFound = true;
            } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                .parseDouble(rangesSplitted[2])) {
                dTotalError += Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                dLineError += Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                if (dLineError > maxError) {
                    maxError = dLineError;
                }
                if (dLineError < minError) {
                    minError = dLineError;
                }
                errorFound = true;
            }
        }
    }

```

```

137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181

```

```

    }
}

if (((rangesSplitted[0].equals("SUMMER")) || (rangesSplitted[0].equals("WINTER"))
    && (rangesSplitted[3].equals("TEMPER.")))) {
    int iSeason = Integer.parseInt(IotDataSplitted2[0].substring(5, 7));

    // checks if SUMMER
    if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
        if (Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
            && Double.parseDouble(IotDataSplitted2[3]) <= Double
            .parseDouble(rangesSplitted[2])) {
            IotData3.add(IotData.get(i));
            // its already on the list no need to add it twice
            break;
        } else {
            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
            if (Double.parseDouble(IotDataSplitted2[3]) < Double
                .parseDouble(rangesSplitted[1])) {
                dTotalError += Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                dLineError += Double.parseDouble(rangesSplitted[1])
                    - Double.parseDouble(IotDataSplitted2[3]);
                TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                if (dLineError > maxError) {
                    maxError = dLineError;
                }
                if (dLineError < minError) {
                    minError = dLineError;
                }
                errorFound = true;
            } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
                .parseDouble(rangesSplitted[2])) {
                dTotalError += Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                dLineError += Double.parseDouble(IotDataSplitted2[3])
                    - Double.parseDouble(rangesSplitted[2]);
                TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
                // break;
                if (dLineError > maxError) {
                    maxError = dLineError;
                }
                if (dLineError < minError) {

```



```

184         errorFound = true;
185     }
186 }
187 }
188 }
189 // checks if WINTER
190 else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
191     if (Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
192         && Double.parseDouble(IotDataSplitted2[3]) <= Double
193             .parseDouble(rangesSplitted[2])) {
194         IotData3.add(IotData.get(i));
195         // its already on the list no need to add it twice
196         break;
197     } else {
198
199         if (Double.parseDouble(IotDataSplitted2[3]) < Double
200             .parseDouble(rangesSplitted[1])) {
201             dTotalError += Double.parseDouble(rangesSplitted[1])
202                 - Double.parseDouble(IotDataSplitted2[3]);
203             dLineError += Double.parseDouble(rangesSplitted[1])
204                 - Double.parseDouble(IotDataSplitted2[3]);
205             TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
206             // break;
207             if (dLineError > maxError) {
208                 maxError = dLineError;
209             }
210             if (dLineError < minError) {
211                 minError = dLineError;
212             }
213             errorFound = true;
214
215         } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
216             .parseDouble(rangesSplitted[2])) {
217             dTotalError += Double.parseDouble(IotDataSplitted2[3])
218                 - Double.parseDouble(rangesSplitted[2]);
219             dLineError += Double.parseDouble(IotDataSplitted2[3])
220                 - Double.parseDouble(rangesSplitted[2]);
221             TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
222             if (dLineError > maxError) {
223                 maxError = dLineError;
224             }
225             if (dLineError < minError) {
226                 minError = dLineError;
227             }
228             errorFound = true;
229

```

```

230     }
231 }
232 }
233 }
234 }
235 }
236 }
237 }
238 if (IotDataSplitted2[2].startsWith("LS")) // gets LS for light sensor
239 {
240     if (rangesSplitted[3].equals("LIGHT")) {
241         if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
242             .parseDouble(rangesSplitted[0].substring(0, 2))
243             && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
244                 .parseDouble(rangesSplitted[0].substring(6, 8)))
245             && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
246             && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
247             IotData3.add(IotData.get(i));
248             // its already on the list no need to add it twice
249             break;
250         } else {
251             // assigns TOTAL ERROR AND CURRENT RECORD ERROR
252             if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
253                 dTotalError += Double.parseDouble(rangesSplitted[1])
254                     - Double.parseDouble(IotDataSplitted2[3]);
255                 dLineError += Double.parseDouble(rangesSplitted[1])
256                     - Double.parseDouble(IotDataSplitted2[3]);
257                 TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
258                 // break;
259                 if (dLineError > maxError) {
260                     maxError = dLineError;
261                 }
262                 if (dLineError < minError) {
263                     minError = dLineError;
264                 }
265                 errorFound = true;
266
267             } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
268                 .parseDouble(rangesSplitted[2])) {
269                 dTotalError += Double.parseDouble(IotDataSplitted2[3])
270                     - Double.parseDouble(rangesSplitted[2]);
271                 dLineError += Double.parseDouble(IotDataSplitted2[3])
272                     - Double.parseDouble(rangesSplitted[2]);
273                 TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);

```

```

275         if (dLineError > maxError) {
276             maxError = dLineError;
277         }
278         if (dLineError < minError) {
279             minError = dLineError;
280         }
281         errorFound = true;
282     }
283 }
284 }
285 }
286
287 if (rangesSplitted[3].equals("SUNNY")) {
288     if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
289         .parseDouble(rangesSplitted[0].substring(0, 2))
290         && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
291             .parseDouble(rangesSplitted[0].substring(6, 8)))
292         && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
293         && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
294         IotData3.add(IotData.get(i));
295         // its already on the list no need to add it twice
296         break;
297     } else {
298         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
299         if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
300             dTotalError += Double.parseDouble(rangesSplitted[1])
301                 - Double.parseDouble(IotDataSplitted2[3]);
302             dLineError += Double.parseDouble(rangesSplitted[1])
303                 - Double.parseDouble(IotDataSplitted2[3]);
304             TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
305             if (dLineError > maxError) {
306                 maxError = dLineError;
307             }
308             if (dLineError < minError) {
309                 minError = dLineError;
310             }
311             errorFound = true;
312         } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
313             .parseDouble(rangesSplitted[2])) {
314             dTotalError += Double.parseDouble(IotDataSplitted2[3])
315                 - Double.parseDouble(rangesSplitted[2]);
316             dLineError += Double.parseDouble(IotDataSplitted2[3])
317                 - Double.parseDouble(rangesSplitted[2]);
318             TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
319

```

```

320         if (dLineError > maxError) {
321             maxError = dLineError;
322         }
323         if (dLineError < minError) {
324             minError = dLineError;
325         }
326         errorFound = true;
327     }
328 }
329 }
330 }
331
332 if (rangesSplitted[3].equals("CLOUDY")) {
333     if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
334         .parseDouble(rangesSplitted[0].substring(0, 2))
335         && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
336             .parseDouble(rangesSplitted[0].substring(6, 8)))
337         && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
338         && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
339         IotData3.add(IotData.get(i));
340         // its already on the list no need to add it twice
341         break;
342     } else {
343         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
344         if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
345             dTotalError += Double.parseDouble(rangesSplitted[1])
346                 - Double.parseDouble(IotDataSplitted2[3]);
347             dLineError += Double.parseDouble(rangesSplitted[1])
348                 - Double.parseDouble(IotDataSplitted2[3]);
349             TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
350             if (dLineError > maxError) {
351                 maxError = dLineError;
352             }
353             if (dLineError < minError) {
354                 minError = dLineError;
355             }
356             errorFound = true;
357         } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
358             .parseDouble(rangesSplitted[2])) {
359             dTotalError += Double.parseDouble(IotDataSplitted2[3])
360                 - Double.parseDouble(rangesSplitted[2]);
361             dLineError += Double.parseDouble(IotDataSplitted2[3])
362                 - Double.parseDouble(rangesSplitted[2]);
363             TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
364

```

```

365         if (dLineError > maxError) {
366             maxError = dLineError;
367         }
368         if (dLineError < minError) {
369             minError = dLineError;
370         }
371         errorFound = true;
372     }
373 }
374 }
375 }
376
377 if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
378     if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
379         .parseDouble(rangesSplitted[0].substring(0, 2))
380         && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
381             .parseDouble(rangesSplitted[0].substring(6, 8)))
382         && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
383         && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
384         IotData3.add(IotData.get(i));
385         break; // its already on the list no need to add it twice
386     } else {
387         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
388         if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
389             dTotalError += Double.parseDouble(rangesSplitted[1])
390                 - Double.parseDouble(IotDataSplitted2[3]);
391             dLineError += Double.parseDouble(rangesSplitted[1])
392                 - Double.parseDouble(IotDataSplitted2[3]);
393             TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
394             if (dLineError > maxError) {
395                 maxError = dLineError;
396             }
397             if (dLineError < minError) {
398                 minError = dLineError;
399             }
400             errorFound = true;
401         } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
402             .parseDouble(rangesSplitted[2])) {
403             dTotalError += Double.parseDouble(IotDataSplitted2[3])
404                 - Double.parseDouble(rangesSplitted[2]);
405             dLineError += Double.parseDouble(IotDataSplitted2[3])
406                 - Double.parseDouble(rangesSplitted[2]);
407             TempIotData.add(makeDataLine(IotData.get(i)) + "| Error: " + dLineError);
408             if (dLineError > maxError) {
409                 maxError = dLineError;
410             }
411             if (dLineError < minError) {
412                 minError = dLineError;
413             }
414             errorFound = true;
415         }
416     }
417 }
418 }
419 }
420
421 }
422
423 }
424
425 if (errorFound == true) {
426     errorCounter = errorCounter + 1;
427     errorFound = false;
428 }
429
430 }
431 } catch (Exception ex) {
432     System.out
433         .println("error first(comparison data with meta-range rules): " + errorLine + ", error msg: " + ex);
434 }
435
436 // save formalized max error
437 formaliseMaxError = maxError;
438
439 dTotalError = dTotalError * 10;
440
441 DecimalFormat df = new DecimalFormat("#.###");
442 df.setRoundingMode(RoundingMode.CEILING);
443
444 String lblError1 = "Total Error: " + Math.round((dTotalError) / formaliseMaxError);
445 String label60 = "Max Error: " + df.format((maxError) * 100 / formaliseMaxError) + "%";
446 String label59 = "Min Error: " + df.format((minError) * 100 / formaliseMaxError) + "%";
447
448 Double averageErrorTotal = Double.parseDouble(df.format((dTotalError / IotData.size())));
449 String label71 = "Total Average Error: " + df.format((averageErrorTotal * 100) / formaliseMaxError) + "%";
450
451 TempIotData.clear();
452 IotData3.clear();
453
454

```

```

455 System.out.println(lblError1);
456 System.out.println(label60);
457 System.out.println(label59);
458 // System.out.println(label58);
459 System.out.println(label71);
460 // *****second run*****
461
462 Double meansSum = 0.0;
463 maxError = 0;
464 minError = 100;
465 dTotalError = 0;
466 dTempError = 0;
467
468 try {
469
470     for (int i = 0; i < IotData.size(); i++) {
471         for (int k = 0; k < listRanges.length; k++) {
472             String[] rangesSplitted = listRanges[k];
473
474             errorLine = makeDataLine(IotData.get(i));
475
476             String[] IotDataSplitted2 = IotData.get(i);
477
478             if (IotDataSplitted2[2].startsWith("T")) // gets T for temperatures
479             {
480                 // checks time and temperature and if season is empty
481                 if (!(rangesSplitted[0].equals("SUMMER")) || !(rangesSplitted[0].equals("WINTER")))
482                     && (rangesSplitted[3].equals("TEMPER.")) {
483                     if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
484                         .parseDouble(rangesSplitted[0].substring(0, 2))
485                         && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
486                         .parseDouble(rangesSplitted[0].substring(6, 8)))
487                         && Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
488                         && Double.parseDouble(IotDataSplitted2[3]) <= Double
489                         .parseDouble(rangesSplitted[2])) {
490                         IotData3.add(IotData.get(i));
491                         break; // its already on the list no need to add it twice
492                     } else {
493                         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
494                         if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
495                             TempIotData.add(makeDataLine(IotData.get(i)));
496                             dTotalError += Double.parseDouble(rangesSplitted[1])
497                                 - Double.parseDouble(IotDataSplitted2[3]);
498                             dTempError = Double.parseDouble(rangesSplitted[1])
499                                 - Double.parseDouble(IotDataSplitted2[3]);
500                         } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
501                             .parseDouble(rangesSplitted[2])) {
502                             TempIotData.add(makeDataLine(IotData.get(i)));
503                             dTotalError += Double.parseDouble(IotDataSplitted2[3])
504                                 - Double.parseDouble(rangesSplitted[2]);
505                             dTempError = Double.parseDouble(IotDataSplitted2[3])
506                                 - Double.parseDouble(rangesSplitted[2]);
507                             break;
508                         }
509                     }
510                 }
511             }
512         }
513     }
514     if ((rangesSplitted[0].equals("SUMMER")) || (rangesSplitted[0].equals("WINTER")))
515         && (rangesSplitted[3].equals("TEMPER.")) {
516         int iSeason = Integer.parseInt(IotDataSplitted2[0].substring(5, 7));
517
518         // checks if SUMMER
519         if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
520             if (Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
521                 && Double.parseDouble(IotDataSplitted2[3]) <= Double
522                 .parseDouble(rangesSplitted[2])) {
523                 IotData3.add(IotData.get(i));
524                 break; // its already on the list no need to add it twice
525             } else {
526                 // assigns TOTAL ERROR AND CURRENT RECORD ERROR
527                 if (Double.parseDouble(IotDataSplitted2[3]) < Double
528                     .parseDouble(rangesSplitted[1])) {
529                     TempIotData.add(makeDataLine(IotData.get(i)));
530                     dTotalError += Double.parseDouble(rangesSplitted[1])
531                         - Double.parseDouble(IotDataSplitted2[3]);
532                     dTempError = Double.parseDouble(rangesSplitted[1])
533                         - Double.parseDouble(IotDataSplitted2[3]);
534                     break;
535                 } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
536                     .parseDouble(rangesSplitted[2])) {
537                     TempIotData.add(makeDataLine(IotData.get(i)));
538                     dTotalError += Double.parseDouble(IotDataSplitted2[3])
539                         - Double.parseDouble(rangesSplitted[2]);
540                     dTempError = Double.parseDouble(IotDataSplitted2[3])
541                         - Double.parseDouble(rangesSplitted[2]);
542                     break;
543                 }
544             }
545         }
546     }

```



```

545     }
546     // checks if WINTER
547     else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
548         if (Double.parseDouble(IotDataSplitted2[3]) >= Double.parseDouble(rangesSplitted[1])
549             && Double.parseDouble(IotDataSplitted2[3]) <= Double
550                 .parseDouble(rangesSplitted[2])) {
551             IotData3.add(IotData.get(i));
552             break; // its already on the list no need to add it twice
553         } else {
554             // assigns TOTAL ERROR AND CURRENT RECORD ERROR
555             if (Double.parseDouble(IotDataSplitted2[3]) < Double
556                 .parseDouble(rangesSplitted[1])) {
557                 TempIotData.add(makeDataLine(IotData.get(i)));
558                 dTotalError += Double.parseDouble(rangesSplitted[1])
559                     - Double.parseDouble(IotDataSplitted2[3]);
560                 dTempError = Double.parseDouble(rangesSplitted[1])
561                     - Double.parseDouble(IotDataSplitted2[3]);
562                 break;
563             } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
564                 .parseDouble(rangesSplitted[2])) {
565                 TempIotData.add(makeDataLine(IotData.get(i)));
566                 dTotalError += Double.parseDouble(IotDataSplitted2[3])
567                     - Double.parseDouble(rangesSplitted[2]);
568                 dTempError = Double.parseDouble(IotDataSplitted2[3])
569                     - Double.parseDouble(rangesSplitted[2]);
570                 break;
571             }
572         }
573     }
574 }
575 }
576 }
577 }
578 }
579 if (IotDataSplitted2[2].startsWith("LS")) // gets LS for light sensor
580 {
581     if (rangesSplitted[3].equals("LIGHT")) {
582         if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
583             .parseDouble(rangesSplitted[0].substring(0, 2))
584             && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
585                 .parseDouble(rangesSplitted[0].substring(6, 8)))
586             && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
587             && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
588             IotData3.add(IotData.get(i));
589             break; // its already on the list no need to add it twice
590         } else {
591             // assigns TOTAL ERROR AND CURRENT RECORD ERROR
592             if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
593                 TempIotData.add(makeDataLine(IotData.get(i)));
594                 dTotalError += Double.parseDouble(rangesSplitted[1])
595                     - Double.parseDouble(IotDataSplitted2[3]);
596                 dTempError = Double.parseDouble(rangesSplitted[1])
597                     - Double.parseDouble(IotDataSplitted2[3]);
598                 break;
599             } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
600                 .parseDouble(rangesSplitted[2])) {
601                 TempIotData.add(makeDataLine(IotData.get(i)));
602                 dTotalError += Double.parseDouble(IotDataSplitted2[3])
603                     - Double.parseDouble(rangesSplitted[2]);
604                 dTempError = Double.parseDouble(IotDataSplitted2[3])
605                     - Double.parseDouble(rangesSplitted[2]);
606                 break;
607             }
608         }
609     }
610 }
611 if (rangesSplitted[3].equals("SUNNY")) {
612     if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
613         .parseDouble(rangesSplitted[0].substring(0, 2))
614         && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
615             .parseDouble(rangesSplitted[0].substring(6, 8)))
616         && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
617         && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
618         IotData3.add(IotData.get(i));
619         break; // its already on the list no need to add it twice
620     } else {
621         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
622         if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
623             TempIotData.add(makeDataLine(IotData.get(i)));
624             dTotalError += Double.parseDouble(rangesSplitted[1])
625                 - Double.parseDouble(IotDataSplitted2[3]);
626             dTempError = Double.parseDouble(rangesSplitted[1])
627                 - Double.parseDouble(IotDataSplitted2[3]);
628             break;
629         } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
630             .parseDouble(rangesSplitted[2])) {
631             TempIotData.add(makeDataLine(IotData.get(i)));
632             dTotalError += Double.parseDouble(IotDataSplitted2[3])
633                 - Double.parseDouble(rangesSplitted[2]);
634             dTempError = Double.parseDouble(IotDataSplitted2[3])

```

```

635         - Double.parseDouble(rangesSplitted[2]);
636         break;
637     }
638 }
639
640
641 if (rangesSplitted[3].equals("CLOUDY")) {
642     if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
643         .parseDouble(rangesSplitted[0].substring(0, 2))
644         && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
645             .parseDouble(rangesSplitted[0].substring(6, 8)))
646         && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
647         && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
648         IotData3.add(IotData.get(i));
649         break; // its already on the list no need to add it twice
650     } else {
651         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
652         if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
653             TempIotData.add(makeDataLine(IotData.get(i)));
654             dTotalError += Double.parseDouble(rangesSplitted[1])
655                 - Double.parseDouble(IotDataSplitted2[3]);
656             dTempError = Double.parseDouble(rangesSplitted[1])
657                 - Double.parseDouble(IotDataSplitted2[3]);
658             break;
659         } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
660             .parseDouble(rangesSplitted[2])) {
661             TempIotData.add(makeDataLine(IotData.get(i)));
662             dTotalError += Double.parseDouble(IotDataSplitted2[3])
663                 - Double.parseDouble(rangesSplitted[2]);
664             dTempError = Double.parseDouble(IotDataSplitted2[3])
665                 - Double.parseDouble(rangesSplitted[2]);
666             break;
667         }
668     }
669 }
670
671 if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
672     if ((Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) >= Double
673         .parseDouble(rangesSplitted[0].substring(0, 2))
674         && Double.parseDouble(IotDataSplitted2[0].substring(11, 13)) <= Double
675             .parseDouble(rangesSplitted[0].substring(6, 8)))
676         && Integer.parseInt(IotDataSplitted2[3]) >= Integer.parseInt(rangesSplitted[1])
677         && Integer.parseInt(IotDataSplitted2[3]) <= Integer.parseInt(rangesSplitted[2])) {
678         IotData3.add(IotData.get(i));
679         break; // its already on the list no need to add it twice
680     } else {
681         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
682         if (Double.parseDouble(IotDataSplitted2[3]) < Double.parseDouble(rangesSplitted[1])) {
683             TempIotData.add(makeDataLine(IotData.get(i)));
684             dTotalError += Double.parseDouble(rangesSplitted[1])
685                 - Double.parseDouble(IotDataSplitted2[3]);
686             dTempError = Double.parseDouble(rangesSplitted[1])
687                 - Double.parseDouble(IotDataSplitted2[3]);
688             break;
689         } else if (Double.parseDouble(IotDataSplitted2[3]) > Double
690             .parseDouble(rangesSplitted[2])) {
691             TempIotData.add(makeDataLine(IotData.get(i)));
692             dTotalError += Double.parseDouble(IotDataSplitted2[3])
693                 - Double.parseDouble(rangesSplitted[2]);
694             dTempError = Double.parseDouble(IotDataSplitted2[3])
695                 - Double.parseDouble(rangesSplitted[2]);
696             break;
697         }
698     }
699 }
700
701 }
702
703 }
704
705 meansSum = meansSum + Math.pow(dLineError - averageErrorTotal, 2);
706
707 }
708 } catch (Exception e) {
709     System.out.println(
710         "error second(comparison data with meta-range rules): " + errorLine + " , Error message: " + e);
711     System.out.println("Number of line:\t" + e.getStackTrace()[2].getLineNumber());
712 }
713
714 double standardDeviation = Math.sqrt(meansSum / IotData.size());

```

```

716 String label57 = "Standard Deviation: " + df.format((standardDeviation) * 100 / formaliseMaxError) + "%";
717 System.out.println(label57);
718
719 IotData3.clear(); // clears the entire list
720 TempPlotData.clear();
721
722 try {
723     FileWriter myWriter = new FileWriter(pathNoIFTTresults);
724     myWriter.write(lblError1);
725     myWriter.write("\n");
726     myWriter.write(label60);
727     myWriter.write("\n");
728
729     myWriter.write(label59);
730     myWriter.write("\n");
731
732     // System.out.println(label58);
733     myWriter.write(label71);
734     myWriter.write("\n");
735
736     myWriter.write(label57);
737     myWriter.write("\n");
738     myWriter.close();
739
740 } catch (Exception e) {
741     System.out.println("Unable to write data for noIFTT.");
742 }
743
744 return formaliseMaxError;
745 } // end of comparison imported data and meta rule ranges
746
747
748
749

```

EnergyPlanner.java


```

1  /*
2      This file implements the Energy Planner algorithm.
3      Copyright (C) 2021 Antonis Vasileiou
4      This program is free software: you can redistribute it and/or modify
5      it under the terms of the GNU General Public License as published by
6      the Free Software Foundation, either version 3 of the License, or
7      (at your option) any later version.
8      This program is distributed in the hope that it will be useful,
9      but WITHOUT ANY WARRANTY; without even the implied warranty of
10     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
11     GNU General Public License for more details.
12     You should have received a copy of the GNU General Public License
13     along with this program. If not, see https://www.gnu.org/licenses/gpl-3.0.html.
14 */
15
16 package iotprojectpackage;
17
18 import java.io.BufferedReader;
19 import java.io.DataOutputStream;
20 import java.io.FileWriter;
21 import java.io.InputStreamReader;
22 import java.math.RoundingMode;
23 import java.net.HttpURLConnection;
24 import java.net.URL;
25 import java.nio.charset.StandardCharsets;
26 import java.text.DecimalFormat;
27 import java.text.ParseException;
28 import java.text.SimpleDateFormat;
29 import java.time.ZoneId;
30 import java.util.ArrayList;
31 import java.util.Date;
32 import java.util.Random;
33 import java.text.DateFormatSymbols;
34
35 import com.google.gson.Gson;
36 import com.google.gson.JsonArray;
37 import com.google.gson.JsonObject;
38 import com.google.gson.JsonParser;
39
40 public class EnergyPlanner {
41
42     private String errorLine; // global
43
44     private SimpleDateFormat simpleFormatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
45     private Date dtTemp;
46     private double contiguous ms = 0;
47
48     private String monthlyKWH_listAssignment = "original"; // to help identify allowed kwh list->original, 1%, 5%
49     // 20%
50     private ArrayList<String[]> copiediotdata;
51     private ArrayList<String[]> EnergyPlannerData;
52     private ArrayList<String[]> copiediotdata3;
53     private ArrayList<String[]> MRTrules;
54     private String kwhpermonth = "";
55     private Boolean solutionsFound;
56     private ArrayList<String[]> recursionResults = new ArrayList<String[]>();
57     private double public_ms = 0;
58     private int recursiveIterator = 0;
59     private String urlString = "http://10.16.30.73/api/update-results";
60     private double totalaverageerror = 0.0;
61     private double budgeterror = 0.0;
62     private double convenienceerror = 0.0;
63     private double deviation = 0.0;
64     private double totalkwh = 0.0;
65     private String listRanges[][];
66
67     public EnergyPlanner() {
68         solutionsFound = false;
69     }
70
71     public void ArrayList_fromString_toStringArray(ArrayList<String> list_from, ArrayList<String[]> list_to) {
72         for (int i = 0; i < list_from.size(); i++) {
73             String temp[] = list_from.get(i).split("\\|");
74             list_to.add(temp);
75         }
76     }
77
78     // call in line 1313
79     public void updateOpenhabRules(String type, String state) {
80         String geturl = "http://10.16.30.73/api/get-openhabrules";
81         String posturl = "http://10.16.30.73/api/update-openhabrules";
82         String postdata = "";
83         try {
84             StringBuilder result = new StringBuilder();
85             URL url = new URL(geturl);
86             HttpURLConnection conn = (HttpURLConnection) url.openConnection();
87             conn.setRequestMethod("GET");
88             conn.connect();

```



```

92     int responsecode = conn.getResponseCode();
93     if (responsecode != 200) {
94         throw new RuntimeException("HttpStatusCode: " + responsecode);
95     } else {
96         BufferedReader br = new BufferedReader(new InputStreamReader(conn.getInputStream()));
97         String line;
98         while ((line = br.readLine()) != null) {
99             result.append(line);
100         }
101     }
102     br.close();
103
104     String temp = result.toString();
105
106     JSONArray array = (JSONArray) JsonParser.parseString(temp);
107
108     int size = array.size();
109
110     if (type.equals("Light")) {
111
112         for (int i = 0; i < size; i++) {
113
114             JSONObject obj = array.get(i).getAsJsonObject();
115
116             if (obj.get("category").getString().equals("Temperature")) {
117                 String state_value = obj.get("state").getString().trim();
118                 String[] str = state_value.split("\\s+");
119                 obj.remove("state");
120                 obj.addProperty("state", str[0] + " \u00b0C"); // This is to handle the Celcius sign
121             }
122
123             if (obj.get("category").getString().equals("Light")) {
124
125                 obj.remove("stateONOFF");
126                 obj.addProperty("stateONOFF", state);
127             }
128
129             array.set(i, obj);
130
131         }
132
133         postdata = array.toString();
134         String urlParameters = "rules=" + postdata;
135         byte[] postData = urlParameters.getBytes(StandardCharsets.UTF_8);
136
137         int postDataLength = postData.length;
138         url = new URL(posturl);
139         conn = (HttpURLConnection) url.openConnection();
140
141         conn.setDoOutput(true);
142         conn.setRequestMethod("POST");
143         conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
144         conn.setRequestProperty("Charset", "UTF-8");
145         conn.setRequestProperty("Content-Length", Integer.toString(postDataLength));
146         conn.setUseCaches(false);
147
148         conn.connect();
149
150         DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
151         wr.write(postData);
152         conn.getResponseCode();
153
154     } // End of If
155     else if (type.equals("Temperature")) {
156         for (int i = 0; i < size; i++) {
157
158             JSONObject obj = array.get(i).getAsJsonObject();
159
160             if (obj.get("category").getString().equals("Temperature")) {
161                 String state_value = obj.get("state").getString().trim();
162                 String[] str = state_value.split("\\s+");
163                 obj.remove("state");
164                 obj.addProperty("state", str[0] + " \u00b0C"); // This is to handle the Celcius sign
165                 obj.remove("stateONOFF");
166                 obj.addProperty("stateONOFF", state);
167             }
168
169             array.set(i, obj);
170
171         }
172
173         postdata = array.toString();
174         String urlParameters = "rules=" + postdata;
175         byte[] postData = urlParameters.getBytes(StandardCharsets.UTF_8);
176         int postDataLength = postData.length;
177         url = new URL(posturl);
178         conn = (HttpURLConnection) url.openConnection();
179
180         conn.setDoOutput(true);

```

```

182         conn.setRequestMethod("POST");
183         conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
184         conn.setRequestProperty("Charset", "UTF-8");
185         conn.setRequestProperty("Content-Length", Integer.toString(postDataLength));
186         conn.setUseCaches(false);
187
188         conn.connect();
189
190         DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
191         wr.write(postData);
192         conn.getResponseCode();
193     }
194 }
195
196 } catch (Exception e) {
197     e.printStackTrace();
198     System.out.println(e.getMessage());
199 }
200
201 }
202
203 // converts string to double
204 public double toDouble(String str) {
205     return Double.parseDouble(str);
206 }
207
208 // converts string to double
209 public int toInt(String str) {
210     return Integer.parseInt(str);
211 }
212
213 // returns hour from record datetime
214 public int getHour(String date) {
215     return toInt(date.substring(11, 13));
216 }
217
218 // returns month from record datetime
219 public int getMonth(String date) {
220     return toInt(date.substring(5, 7));
221 }
222
223 // returns the hour the rule starts
224 public int ruleHourFrom(String ruletime) {
225     return toInt(ruletime.substring(0, 2));
226 }
227
228 // returns the hour the rule ends
229 public int ruleHourTo(String ruletime) {
230     return toInt(ruletime.substring(6, 8));
231 }
232
233 public void postData() {
234     try {
235         String urlParameters = "ep_kWh=" + totalkwh + "&ep_totalError=" + totalaverageerror
236             + "&ep_convenienceError=" + convinienceerror + "&ep_budgetError=" + budgeterror
237             + "&ep_standardDeviation=" + deviation + "&kwh_per_month=" + kwhpermonth;
238         byte[] postData = urlParameters.getBytes(StandardCharsets.UTF_8);
239         int postDataLength = postData.length;
240
241         URL url = new URL(urlString);
242         HttpURLConnection conn = (HttpURLConnection) url.openConnection();
243
244         conn.setDoOutput(true);
245         conn.setRequestMethod("POST");
246         conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
247         conn.setRequestProperty("Charset", "UTF-8");
248         conn.setRequestProperty("Content-Length", Integer.toString(postDataLength));
249         conn.setUseCaches(false);
250
251         conn.connect();
252
253         DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
254         wr.write(postData);
255         conn.getResponseCode();
256     } catch (Exception e) {

```

```

272         for (StackTraceElement s : e.getStackTrace())
273             System.out.println(s);
274     }
275 }
276
277 public String makeDataLine(String[] array) {
278     String s = "";
279     String seperator = "|";
280     int count = 0;
281     int length = array.length;
282
283     for (String temp : array) {
284         if (count == length - 1) {
285             s = s.concat(temp);
286         } else {
287             s = s.concat(temp);
288             s = s.concat(seperator);
289         }
290         count++;
291     }
292
293     return s;
294 }
295
296 public static String[] addX(String myarray[], String ele) {
297
298     int n = myarray.length;
299     String newArray[] = new String[n + 1];
300     // copy original array into new array
301     for (int i = 0; i < n; i++)
302         newArray[i] = myarray[i];
303
304     // add element to the new array
305     newArray[n] = ele;
306
307     return newArray;
308 }
309
310 private double calculateKWH3(ArrayList<String[]> sRecordsHourly) {
311     ArrayList<String[]> copiediotdatakwh = new ArrayList<String[]>(sRecordsHourly);
312     ArrayList<Integer> removedMetaRules = new ArrayList<Integer>();
313     boolean randomRule[] = new boolean[MRTrules.size()];
314     Random rand = new Random();
315     int numrules = MRTrules.size();
316     int random;
317
318     Boolean startALLzeros = false;
319     Boolean startALLrandom = false;
320     int numberOfLoops = 3;
321
322     if (removedMetaRules.size() == 0) {
323         if (startALLzeros == true) {
324             for (int d = 0; d < randomRule.length; d++) {
325                 randomRule[d] = false;
326                 removedMetaRules.add(d);
327             }
328         } else if (startALLrandom == true) {
329             for (int d = 0; d < randomRule.length; d++) {
330                 // generates random number from 0 to 1
331                 random = rand.nextInt(2);
332                 if (random == 0) {
333                     randomRule[d] = false;
334                     removedMetaRules.add(d);
335                 } else {
336                     randomRule[d] = true;
337                 }
338             }
339         } else {
340             for (int d = 0; d < randomRule.length; d++) {
341                 randomRule[d] = true;
342             }
343         }
344
345         for (int i = 0; i < numberOfLoops; i++) {
346             random = rand.nextInt(numrules);
347             // random here is the rule to be removed
348             if (randomRule[random]) {
349                 randomRule[random] = false;
350                 removedMetaRules.add(random);
351             } else {
352                 randomRule[random] = true;
353                 removedMetaRules.remove(Integer.valueOf(random));
354             }
355         }
356
357         if (solutionsFound == false) {
358
359             for (int i = 0; i < numberOfLoops; i++) {
360                 random = rand.nextInt(numrules);
361                 // random here is the rule to be removed

```

```

362         if (randomRule[random]) {
363             randomRule[random] = false;
364             removedMetaRules.add(random);
365         } else {
366             randomRule[random] = true;
367             removedMetaRules.remove(Integer.valueOf(random));
368         }
369     }
370 } else if (solutionsFound == true) {
371     for (int d = 0; d < randomRule.length; d++) {
372         randomRule[d] = true;
373     }
374
375     for (int i = 0; i < removedMetaRules.size(); i++) {
376         randomRule[removedMetaRules.get(i)] = false;
377     }
378 }
379
380 int IFTTTconstraintsApplied = 0;
381 double milliWatts = 0;
382 Date dt1 = dtTemp;
383 Date dt2;
384 double span;
385 double ms = 0;
386 double localContiguous_ms = contiguous_ms;
387 ArrayList<Double> monthlyKWH = new ArrayList<Double>();
388
389 if (monthlyKWH_listAssignment.equals("original")) {
390     monthlyKWH = new ArrayList<Double>(); // 0.24
391     Double[] d = new Double[] { 1.04, 0.71, 0.33, 0.19, 0.24, 0.28, 0.33, 0.43, 0.28, 0.34, 0.28, 0.57 };
392     for (int i = 0; i < d.length; i++)
393         monthlyKWH.add(d[i]);
394 } else if (monthlyKWH_listAssignment.equals("1")) {
395     monthlyKWH = new ArrayList<Double>();
396     Double[] d = new Double[] { 1.03, 0.70, 0.33, 0.19, 0.23, 0.28, 0.33, 0.42, 0.28, 0.23, 0.28, 0.56 };
397     for (int i = 0; i < d.length; i++)
398         monthlyKWH.add(d[i]);
399 } else if (monthlyKWH_listAssignment.equals("5")) {
400     monthlyKWH = new ArrayList<Double>();
401
402     Double[] d = new Double[] { 0.99, 0.68, 0.32, 0.18, 0.23, 0.27, 0.32, 0.41, 0.27, 0.23, 0.27, 0.54 };
403     for (int i = 0; i < d.length; i++)
404         monthlyKWH.add(d[i]);
405 } else if (monthlyKWH_listAssignment.equals("10")) {
406     monthlyKWH = new ArrayList<Double>();
407
408     Double[] d = new Double[] { 0.94, 0.64, 0.30, 0.17, 0.21, 0.26, 0.30, 0.38, 0.26, 0.21, 0.26, 0.51 };
409     for (int i = 0; i < d.length; i++)
410         monthlyKWH.add(d[i]);
411 } else if (monthlyKWH_listAssignment.equals("20")) {
412     monthlyKWH = new ArrayList<Double>();
413
414     Double[] d = new Double[] { 0.83, 0.57, 0.27, 0.15, 0.19, 0.23, 0.27, 0.34, 0.23, 0.19, 0.23, 0.45 };
415     for (int i = 0; i < d.length; i++)
416         monthlyKWH.add(d[i]);
417 } else if (monthlyKWH_listAssignment.equals("30")) {
418     monthlyKWH = new ArrayList<Double>();
419     Double[] d = new Double[] { 0.73, 0.50, 0.23, 0.13, 0.17, 0.20, 0.23, 0.30, 0.20, 0.17, 0.20, 0.40 };
420     for (int i = 0; i < d.length; i++)
421         monthlyKWH.add(d[i]);
422 } else if (monthlyKWH_listAssignment.equals("40")) {
423     monthlyKWH = new ArrayList<Double>();
424     Double[] d = new Double[] { 0.63, 0.43, 0.20, 0.11, 0.14, 0.17, 0.20, 0.26, 0.17, 0.14, 0.17, 0.34 };
425     for (int i = 0; i < d.length; i++)
426         monthlyKWH.add(d[i]);
427 }
428 try {
429
430     ArrayList<String> listRules = new ArrayList<String>();
431     for (int d = 0; d < randomRule.length; d++) {
432         if (randomRule[d]) {
433
434             // temperature
435             if (listRanges[d][3].equals("TEMPER.")) {
436                 listRules.add(listRanges[d][0] + "|" + "+" + listRanges[d][1] + "|" + "----" + "|" +
437                     + listRanges[d][1] + "|" + "----" + "|" + "----" + "|" + "----" + "|" + "SET TEMPERATURE");
438             }
439             // light
440             if (listRanges[d][3].equals("LIGHT")) {
441                 listRules.add(listRanges[d][0] + "|" + "----" + "|" + "+" + listRanges[d][1] + "|" +
442                     + listRanges[d][1] + "|" + "----" + "|" + "----" + "|" + "----" + "|" + "SET LIGHT");
443             }
444             IFTTTconstraintsApplied++;
445         }
446     }
447
448     for (int i = 0; i < copiediotdatakwh.size(); i++) {
449
450         for (int k = 0; k < listRules.size(); k++) {
451             errorLine = String.join("|", copiediotdatakwh.get(i));

```

```

452
453 String[] copiediotdataSplitted = copiediotdatakwh.get(i);
454 String[] rulesSplitted = listRules.get(k).split("\\|");
455
456 if (copiediotdataSplitted[2].startsWith("T")) // gets T for temperatures
457 {
458     // checks time and temperature and if season,wether,door is empty
459     if ((!rulesSplitted[1].equals("----")) && (rulesSplitted[4].equals("----"))
460         && (rulesSplitted[5].equals("----")) && (rulesSplitted[6].equals("----"))) {
461
462         if ((getHour(copiediotdataSplitted[0]) >= ruleHourFrom(rulesSplitted[0])
463             && getHour(copiediotdataSplitted[0]) <= ruleHourTo(rulesSplitted[0]))) {
464             dt2 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
465             span = Math.abs(dt2.getTime() - dt1.getTime());
466             ms += span;
467             dt1 = dt2;
468
469             localContiguous_ms += span;
470             double secondsTemp = localContiguous_ms / 1000;
471             double hoursTemp = secondsTemp / 3600;
472             double kWhTemp = ((hoursTemp * 320) / 1000) * 24;
473             copiediotdataSplitted = addX(copiediotdataSplitted, String.valueOf(kWhTemp));
474
475             milliWatts = milliWatts + 320;
476             copiediotdataSplitted[3] = rulesSplitted[3];
477             copiediotdatakwh.set(i, copiediotdataSplitted);
478             break; // its already on the list no need to add it twice
479         }
480     }
481 }
482
483 if (copiediotdataSplitted[2].startsWith("LS")) // gets LS for light sensor
484 {
485     // checks LIGHT , and if weather,door is empty
486     if ((!rulesSplitted[2].equals("----")) && (rulesSplitted[5].equals("----"))
487         && (rulesSplitted[6].equals("----"))) {
488         // checks time and light value
489         if ((getHour(copiediotdataSplitted[0]) >= ruleHourFrom(rulesSplitted[0])
490             && getHour(copiediotdataSplitted[0]) <= ruleHourTo(rulesSplitted[0]))) {
491             dt2 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
492             span = Math.abs(dt2.getTime() - dt1.getTime());
493             ms += span;
494             dt1 = dt2;
495
496             localContiguous_ms += span;
497             double secondsTemp = localContiguous_ms / 1000;
498             double hoursTemp = secondsTemp / 3600;
499             double kWhTemp = ((hoursTemp * 320) / 1000) * 24;
500             copiediotdataSplitted = addX(copiediotdataSplitted, String.valueOf(kWhTemp));
501
502             copiediotdataSplitted[3] = rulesSplitted[3];
503             copiediotdatakwh.set(i, copiediotdataSplitted);
504             break;
505         }
506     }
507 } // end of : //checks LIGHT and if weather is empty
508
509 if (k == listRules.size() - 1) {
510     dt1 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
511 }
512
513 String[] copiediotdataSplittedAfterIFTTT = copiediotdatakwh.get(i);
514 if (copiediotdataSplittedAfterIFTTT.length < 8) {
515     copiediotdatakwh.set(i, addX(copiediotdataSplittedAfterIFTTT, "00"));
516 }
517
518 } catch (Exception ex) {
519     System.out.println("First B: " + errorLine + " => Exception: " + ex);
520 }
521
522 double seconds = ms / 1000;
523 double hours = seconds / 3600;
524
525 double kWh = ((hours * 320) / 1000) * 24;
526
527 String[] copiediotdataSplitted_Month = copiediotdatakwh.get(0);
528 Date getMonth;
529 int currentMonth = 0;
530 try {
531     getMonth = simpleFormatter.parse(copiediotdataSplitted_Month[0].substring(0, 18));
532     currentMonth = getMonth.toInstant().atZone(ZoneId.systemDefault()).toLocalDate().getMonthValue();
533 } catch (ParseException e1) {

```

```

542 // TODO Auto-generated catch block
543 e1.printStackTrace();
544 }
545 double allowedMonthly_kWh = toDouble(monthlyKWH.getCurrentMonth - 1).toString()) * 6.5;// 6.5, *7.9;
546
547 // *****SECOND AND LAST
548 // PHASE*****//
549 double dTotalError = 0;
550 double dLineError = 0;
551 double dTotalError_kWh = 0;
552 double maxError = 0;
553 double minError = 100;
554 boolean errorFound = false;
555 int errorSize = 0;
556
557 try {
558
559     for (int i = 0; i < copiediotdatakwh.size(); i++) {
560
561         dLineError = 0;// gets the total record line error in regards of all the ranges
562         for (int k = 0; k < listRanges.length; k++) {
563             errorLine = String.join("|", copiediotdatakwh.get(i));
564
565             String[] copiediotdataSplitted2 = copiediotdatakwh.get(i);
566             // System.out.println(copiediotdatakwh.get(i));
567             String[] rangesSplitted = listRanges[k];
568
569             if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
570             {
571
572                 // checks time and temperature and if season is empty
573                 if (((!rangesSplitted[0].equals("SUMMER")) || (!rangesSplitted[0].equals("WINTER"))))
574                     && (rangesSplitted[3].equals("TEMPER."))) {
575                     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
576                         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
577                         && toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
578                         && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
579                         copiediotdata3.add(copiediotdatakwh.get(i));
580                     }
581                 } else {
582                     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
583                     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
584                         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
585                         && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
586                         dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
587
588                         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
589
590                         if (dLineError > maxError) {
591                             maxError = dLineError;
592                         }
593                         if (dLineError < minError) {
594                             minError = dLineError;
595                         }
596                         errorFound = true;
597                     } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
598                         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
599                         && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
600                         dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
601                         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
602
603                         if (dLineError > maxError) {
604                             maxError = dLineError;
605                         }
606                         if (dLineError < minError) {
607                             minError = dLineError;
608                         }
609                         errorFound = true;
610                     }
611                 }
612             }
613
614             if (((rangesSplitted[0].equals("SUMMER")) || (rangesSplitted[0].equals("WINTER"))))
615                 && (rangesSplitted[3].equals("TEMPER."))) {
616                 int iSeason = getMonth(copiediotdataSplitted2[0]);
617
618                 // checks if SUMMER
619                 if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
620                     if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
621                         && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
622                         copiediotdata3.add(copiediotdatakwh.get(i));
623                     }
624                 } else {
625                     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
626                     if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
627                         dTotalError += toDouble(rangesSplitted[1])
628                             - toDouble(copiediotdataSplitted2[3]);
629                         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
630
631                         if (dLineError > maxError) {

```



```

632         maxError = dLineError;
633     }
634     if (dLineError < minError) {
635         minError = dLineError;
636     }
637     errorFound = true;
638 } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
639     dTotalError += toDouble(copiediotdataSplitted2[3])
640         - toDouble(rangesSplitted[2]);
641     dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
642
643     if (dLineError > maxError) {
644         maxError = dLineError;
645     }
646     if (dLineError < minError) {
647         minError = dLineError;
648     }
649     errorFound = true;
650 }
651 }
652 }
653 // checks if WINTER
654 else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
655     if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
656         && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
657         copiediotdata3.add(copiediotdatakwh.get(i));
658     } else {
659         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
660         if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
661             dTotalError += toDouble(rangesSplitted[1])
662                 - toDouble(copiediotdataSplitted2[3]);
663             dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
664
665             if (dLineError > maxError) {
666                 maxError = dLineError;
667             }
668             if (dLineError < minError) {
669                 minError = dLineError;
670             }
671             errorFound = true;
672         } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
673             dTotalError += toDouble(copiediotdataSplitted2[3])
674                 - toDouble(rangesSplitted[2]);
675             dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
676
677             if (dLineError > maxError) {
678                 maxError = dLineError;
679             }
680             if (dLineError < minError) {
681                 minError = dLineError;
682             }
683             errorFound = true;
684         }
685     }
686 }
687 }
688 }
689 }
690 }
691 if (rangesSplitted[0].equals("kWh")) {
692     if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
693         copiediotdata3.add(copiediotdatakwh.get(i));
694     } else {
695         dTotalError_kWh += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
696         dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
697         dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
698
699         if (dLineError > maxError) {
700             maxError = dLineError;
701         }
702         if (dLineError < minError) {
703             minError = dLineError;
704         }
705         errorFound = true;
706     }
707 }
708 }
709 }
710 }
711 if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
712 {
713     // checks LIGHT
714     if (rangesSplitted[3].equals("LIGHT")) {
715         if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
716             && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
717             && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
718             && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
719             copiediotdata3.add(copiediotdatakwh.get(i));
720         } else {
721

```

```

722 // assigns TOTAL ERROR AND CURRENT RECORD ERROR
723 if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
724     && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
725     && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
726     dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
727     dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
728
729     if (dLineError > maxError) {
730         maxError = dLineError;
731     }
732     if (dLineError < minError) {
733         minError = dLineError;
734     }
735     errorFound = true;
736 } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
737     && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
738     && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
739     dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
740     dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
741
742     if (dLineError > maxError) {
743         maxError = dLineError;
744     }
745     if (dLineError < minError) {
746         minError = dLineError;
747     }
748     errorFound = true;
749 }
750 }
751 } // end of: checks LIGHT
752
753 // checks SUNNY
754 if (rangesSplitted[3].equals("SUNNY")) {
755     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
756         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
757         && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
758         && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
759         copiediotdata3.add(copiediotdatakwh.get(i));
760     }
761 } else {
762     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
763     if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
764         dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
765         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
766
767         if (dLineError > maxError) {
768             maxError = dLineError;
769         }
770         if (dLineError < minError) {
771             minError = dLineError;
772         }
773         errorFound = true;
774     } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
775         dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
776         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
777
778         if (dLineError > maxError) {
779             maxError = dLineError;
780         }
781         if (dLineError < minError) {
782             minError = dLineError;
783         }
784         errorFound = true;
785     }
786 }
787 } // end of :checks SUNNY
788
789 // checks CLOUDY
790 if (rangesSplitted[3].equals("CLOUDY")) {
791     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
792         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
793         && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
794         && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
795         copiediotdata3.add(copiediotdatakwh.get(i));
796     }
797 } else {
798     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
799     if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
800         dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
801         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
802
803         if (dLineError > maxError) {
804             maxError = dLineError;
805         }
806         if (dLineError < minError) {
807             minError = dLineError;
808         }
809         errorFound = true;
810     } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
811         dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);

```



```

812         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
813
814         if (dLineError > maxError) {
815             maxError = dLineError;
816         }
817         if (dLineError < minError) {
818             minError = dLineError;
819         }
820         errorFound = true;
821     }
822 } // end of :checks CLOUDY
823
824 // checks DOOR OPEN-LIGHT
825 if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
826     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
827         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
828         && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
829         && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
830         copiediotdata3.add(copiediotdatakwh.get(i));
831     }
832 } else {
833     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
834     if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
835         dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
836         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
837
838         if (dLineError > maxError) {
839             maxError = dLineError;
840         }
841         if (dLineError < minError) {
842             minError = dLineError;
843         }
844         errorFound = true;
845     } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
846         dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
847         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
848
849         if (dLineError > maxError) {
850             maxError = dLineError;
851         }
852         if (dLineError < minError) {
853             minError = dLineError;
854         }
855         errorFound = true;
856     }
857 }
858 } // end of :checks DOOR OPEN-LIGHT
859
860 if (rangesSplitted[0].equals("kWh")) {
861     if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
862         copiediotdata3.add(copiediotdatakwh.get(i));
863     } else {
864         dTotalError_kWh += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
865         dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
866         dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
867
868         if (dLineError > maxError) {
869             maxError = dLineError;
870         }
871         if (dLineError < minError) {
872             minError = dLineError;
873         }
874         errorFound = true;
875     }
876 } // end of kWh check
877 } // end of LS
878
879 }
880
881 if (errorFound == true) {
882     errorsSize++;
883     errorFound = false;
884 }
885
886 }
887
888 } catch (Exception e) {
889     System.out.println("Second B: " + errorLine + "\tException:\t" + e);
890     e.printStackTrace();
891 }
892
893 // TempIotData.clear();
894 copiediotdata3.clear();
895
896 // *****end of second phase (calculating Error
897 // phase)*****
898
899
900
901

```

```

902 if ((kWh > allowedMonthly_kWh) && (solutionsFound == false)) {
903
904     if (removedMetaRules.size() == numrules) {
905         removedMetaRules.clear();
906     }
907     calculateKWH3(sRecordsHourly);
908     return public_ms;
909
910 } else if (recursiveIterator < 5) {
911     if (recursionResults.size() == 0) // check if result found first time
912     {
913         String temp[] = new String[3];
914         String t = "";
915         boolean first = true;
916         for (int d = 0; d < removedMetaRules.size(); d++) {
917             if (first) {
918                 t = removedMetaRules.get(d).toString();
919                 first = false;
920             } else {
921                 t = t.concat(", " + removedMetaRules.get(d));
922             }
923         }
924         temp[0] = t;
925         temp[1] = String.valueOf(kWh);
926         temp[2] = String.valueOf(dTotalError);
927         recursionResults.add(temp);
928     } else // now check if new result is better than the previous and replace it
929     {
930         String[] splitted = recursionResults.get(0);
931         if (dTotalError < toDouble(splitted[2])) {
932             String temp[] = new String[3];
933             String t = "";
934             boolean first = true;
935             for (int d = 0; d < removedMetaRules.size(); d++) {
936                 if (first) {
937                     t = removedMetaRules.get(d).toString();
938                     first = false;
939                 } else {
940                     t = t.concat(", " + removedMetaRules.get(d));
941                 }
942             }
943             temp[0] = t;
944             temp[1] = String.valueOf(kWh);
945             temp[2] = String.valueOf(dTotalError);
946             // recursionResults.set(0, String.join(", ", removedMetaRules) + "|" +
947
948             // String.valueOf(kWh) + "|" + String.valueOf(dTotalError));
949             recursionResults.set(0, temp);
950         }
951     }
952
953     recursiveIterator++;
954     if (removedMetaRules.size() == numrules) {
955         removedMetaRules.clear();
956     }
957     if (recursiveIterator == 5) {
958         solutionsFound = true;
959         removedMetaRules.clear();
960         String[] rs = recursionResults.get(0);
961         String[] splitted = rs[0].split(",");
962         for (int i = 0; i < splitted.length; i++) {
963             if (splitted[0].length() == 0)
964                 break;
965             removedMetaRules.add(Integer.valueOf(splitted[i]));
966         }
967     }
968     calculateKWH3(sRecordsHourly);
969     return public_ms;
970
971 } else {
972     contiguous_ms = localContiguous_ms;
973     dtTemp = dt1;
974     EnergyPlannerData.addAll(copiediotdatakwh);
975     removedMetaRules.clear();
976     recursiveIterator = 0;
977     recursionResults.clear();
978     solutionsFound = false;
979
980     public_ms = ms;
981     return public_ms;
982 }
983
984 }
985
986 } // END OF CALCULATE KWH METHOD - number of modifications
987
988 public void energyplanner(ArrayList<String[]> data, ArrayList<String[]> MRTrulesOriginal,
989     Double formaliseMaxError) {
990     copiediotdata = new ArrayList<String[]>(data); // clone original dataset list
991     MRTrules = new ArrayList<String[]>(MRTrulesOriginal);
992     listRanges = MRTrules.toArray(new String[0][1]);

```

```

992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081

try {
    dtTemp = simpleFormatter.parse("2013-10-22 02:32:34.409"); // temp date
} catch (Exception e) {
    System.out.println(e);
}

contiguous_ms = 0;
ArrayList<String[]> recordsHourly = new ArrayList<String[]>();
double hourChange = 2;
double total_ms = 0;
int monthchange = 0;
int yearchange = 1998;
DecimalFormat df1 = new DecimalFormat("#.##");
df1.setRoundingMode(RoundingMode.CEILING);
String monthString;
EnergyPlannerData = new ArrayList<String[]>();
copiediotdata3 = new ArrayList<String[]>(); // clone original dataset list
boolean first = true;

try {
    for (int i = 0; i < copiediotdata.size(); i++) {
        // System.out.println(i);
        String[] copiediotdataSplitHour = copiediotdata.get(i);
        int tempHourChange = getHour(copiediotdataSplitHour[0]);
        int tempmonthchange = getMonth(copiediotdataSplitHour[0]);

        if (hourChange == tempHourChange) {
            if (i == copiediotdata.size() - 1) {
                recordsHourly.add(copiediotdata.get(i));
                double hourly_ms = calculateKWH3(recordsHourly);
                total_ms = total_ms + hourly_ms;
                recordsHourly.clear();
            } else {
                recordsHourly.add(copiediotdata.get(i));
            }
        } else {
            hourChange = tempHourChange;
            double hourly_ms = calculateKWH3(recordsHourly);
            total_ms = total_ms + hourly_ms;
            recordsHourly.clear();
            recordsHourly.add(copiediotdata.get(i));
        }
    }

    // this is the calculation for the kwh per month
    if (monthchange != tempmonthchange) {
        monthchange = tempmonthchange;
        // first is first month
        if (first) {
            double seconds1 = total_ms / 1000;
            double hours1 = seconds1 / 3600;
            double kWh1 = ((hours1 * 320) / 1000) * 24;

            kwhpermonth = kwhpermonth.concat(df1.format(kWh1));
            first = false;
        } else {
            double seconds1 = total_ms / 1000;
            double hours1 = seconds1 / 3600;
            double kWh1 = ((hours1 * 320) / 1000) * 24;

            kwhpermonth = kwhpermonth.concat(", " + df1.format(kWh1));
        }
    }
    // END of calculation for the kwh per month
} catch (Exception ex) {
    System.out.println("First A: " + errorLine + " => Exception: " + ex);
    ex.printStackTrace();
}

copiediotdata.clear();

copiediotdata = new ArrayList<String[]>(EnergyPlannerData);
EnergyPlannerData.clear();

DecimalFormat df = new DecimalFormat("#.##");
df.setRoundingMode(RoundingMode.CEILING);

double seconds = total_ms / 1000;
double hours = seconds / 3600;
double kWh = ((hours * 320) / 1000) * 24;
String label154 = "Kilowatt hour (kWh): " + df.format(kWh) + " kWh";
System.out.println(label154);
totalkwh = toDouble(df.format(kWh));
double dTotalError = 0;

```

```

1082 double dLineError = 0;
1083 double dTotalError_kWh = 0;
1084
1085 // *****SECOND AND LAST PHASE*****//
1086
1087 Double maxError = 0.0;
1088 Double minError = 100.0;
1089 Boolean errorFound = false;
1090 int errorsize = 0;
1091
1092 try {
1093     for (int i = 0; i < copiediotdata.size(); i++) {
1094
1095         dLineError = 0; // gets the total record line error in regards of all the ranges
1096         for (int k = 0; k < listRanges.length; k++) {
1097             errorLine = makeDataLine(copiediotdata.get(i));
1098
1099             String[] copiediotdataSplitted2 = copiediotdata.get(i);
1100             String[] rangesSplitted = listRanges[k];
1101
1102             if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
1103             {
1104
1105                 // checks time and temperature and if season is empty
1106                 if (((!rangesSplitted[0].equals("SUMMER")) || (!rangesSplitted[0].equals("WINTER"))))
1107                     && (rangesSplitted[3].equals("TEMPER."))) {
1108                     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1109                         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1110                         && toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
1111                         && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
1112                         //updateOpenhabRules("Temperature", "ON");
1113                         copiediotdata3.add(copiediotdata.get(i));
1114                     } else {
1115
1116                         //updateOpenhabRules("Temperature", "OFF");
1117
1118                         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1119
1120                         if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1121                             && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1122                             && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1123                             dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1124                             dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1125                         }
1126
1127                         if (dLineError > maxError) {
1128                             maxError = dLineError;
1129                         }
1130
1131                         if (dLineError < minError) {
1132                             minError = dLineError;
1133                         }
1134                         errorFound = true;
1135                     } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1136                         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1137                         && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1138                         dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1139                         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1140
1141                         if (dLineError > maxError) {
1142                             maxError = dLineError;
1143                         }
1144                         if (dLineError < minError) {
1145                             minError = dLineError;
1146                         }
1147                         errorFound = true;
1148                     }
1149                 }
1150             }
1151         }
1152     }
1153
1154     if (((rangesSplitted[0].equals("SUMMER")) || (rangesSplitted[0].equals("WINTER"))))
1155         && (rangesSplitted[3].equals("TEMPER."))) {
1156         int iSeason = getMonth(copiediotdataSplitted2[0]);
1157
1158         // checks if SUMMER
1159         if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
1160             if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
1161                 && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
1162                 copiediotdata3.add(copiediotdata.get(i));
1163             } else {
1164                 // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1165                 if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1166                     dTotalError += toDouble(rangesSplitted[1])
1167                         - toDouble(copiediotdataSplitted2[3]);
1168                     dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1169                 }
1170                 if (dLineError > maxError) {
1171                     maxError = dLineError;
1172                 }
1173             }
1174         }
1175     }

```

```

1172     }
1173     if (dLineError < minError) {
1174         minError = dLineError;
1175     }
1176     errorFound = true;
1177 } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1178     dTotalError += toDouble(copiediotdataSplitted2[3])
1179     - toDouble(rangesSplitted[2]);
1180     dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1181
1182     if (dLineError > maxError) {
1183         maxError = dLineError;
1184     }
1185     if (dLineError < minError) {
1186         minError = dLineError;
1187     }
1188     errorFound = true;
1189 }
1190 }
1191 }
1192 // checks if WINTER
1193 else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
1194     if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
1195         && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
1196         copiediotdata3.add(copiediotdata.get(i));
1197     }
1198     } else {
1199         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1200         if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1201             dTotalError += toDouble(rangesSplitted[1])
1202             - toDouble(copiediotdataSplitted2[3]);
1203             dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1204
1205             if (dLineError > maxError) {
1206                 maxError = dLineError;
1207             }
1208             if (dLineError < minError) {
1209                 minError = dLineError;
1210             }
1211             errorFound = true;
1212         } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1213             dTotalError += toDouble(copiediotdataSplitted2[3])
1214             - toDouble(rangesSplitted[2]);
1215             dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1216
1217             if (dLineError > maxError) {
1218                 maxError = dLineError;
1219             }
1220             if (dLineError < minError) {
1221                 minError = dLineError;
1222             }
1223             errorFound = true;
1224         }
1225     }
1226 }
1227 }
1228 }
1229 }
1230 if (rangesSplitted[0].equals("kWh")) {
1231     if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
1232         copiediotdata3.add(copiediotdata.get(i));
1233     } else {
1234         dTotalError_kWh += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
1235         dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
1236         dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
1237
1238         if (dLineError > maxError) {
1239             maxError = dLineError;
1240         }
1241         if (dLineError < minError) {
1242             minError = dLineError;
1243         }
1244         errorFound = true;
1245     }
1246 }
1247 }
1248 }
1249 }
1250 if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
1251 {
1252     // checks LIGHT
1253     if (rangesSplitted[3].equals("LIGHT")) {
1254         if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1255             && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1256             && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1257             && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1258             copiediotdata3.add(copiediotdata.get(i));
1259             //updateOpenhabRules("Light", "ON");
1260         }
1261     } else {

```

```

1262 //updateOpenhabRules("Light", "OFF");
1263 // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1264 if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1265     && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1266     && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1267     dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1268     dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1269
1270     if (dLineError > maxError) {
1271         maxError = dLineError;
1272     }
1273     if (dLineError < minError) {
1274         minError = dLineError;
1275     }
1276     errorFound = true;
1277 } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1278     && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1279     && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1280     dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1281     dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1282
1283     if (dLineError > maxError) {
1284         maxError = dLineError;
1285     }
1286     if (dLineError < minError) {
1287         minError = dLineError;
1288     }
1289     errorFound = true;
1290 }
1291 }
1292 } // end of: checks LIGHT
1293
1294 // checks SUNNY
1295 if (rangesSplitted[3].equals("SUNNY")) {
1296     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1297         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1298         && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1299         && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1300         copiediotdata3.add(copiediotdata.get(i));
1301     }
1302     } else if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1303         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1304
1305         dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1306         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1307
1308         if (dLineError > maxError) {
1309             maxError = dLineError;
1310         }
1311         if (dLineError < minError) {
1312             minError = dLineError;
1313         }
1314         errorFound = true;
1315     } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1316         dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1317         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1318
1319         if (dLineError > maxError) {
1320             maxError = dLineError;
1321         }
1322         if (dLineError < minError) {
1323             minError = dLineError;
1324         }
1325         errorFound = true;
1326     }
1327 }
1328 } // end of :checks SUNNY
1329
1330 // checks CLOUDY
1331 if (rangesSplitted[3].equals("CLOUDY")) {
1332     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1333         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1334         && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1335         && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1336         copiediotdata3.add(copiediotdata.get(i));
1337     }
1338     } else {
1339         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1340         if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1341             dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1342             dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1343
1344             if (dLineError > maxError) {
1345                 maxError = dLineError;
1346             }
1347             if (dLineError < minError) {
1348                 minError = dLineError;
1349             }
1350             errorFound = true;
1351         } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {

```



```

1352         dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1353         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1354
1355         if (dLineError > maxError) {
1356             maxError = dLineError;
1357         }
1358         if (dLineError < minError) {
1359             minError = dLineError;
1360         }
1361         errorFound = true;
1362     }
1363 } // end of :checks CLOUDY
1364
1365 // checks DOOR OPEN-LIGHT
1366 if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
1367     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1368         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1369         && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1370         && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1371         copiediotdata3.add(copiediotdata.get(i));
1372     }
1373 } else {
1374     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1375     if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1376         dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1377         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1378
1379         if (dLineError > maxError) {
1380             maxError = dLineError;
1381         }
1382         if (dLineError < minError) {
1383             minError = dLineError;
1384         }
1385         errorFound = true;
1386     } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1387         dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1388         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1389
1390         if (dLineError > maxError) {
1391             maxError = dLineError;
1392         }
1393         if (dLineError < minError) {
1394             minError = dLineError;
1395         }
1396     }
1397     errorFound = true;
1398 }
1399 } // end of :checks DOOR OPEN-LIGHT
1400
1401 if (rangesSplitted[0].equals("kWh")) {
1402     if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
1403         copiediotdata3.add(copiediotdata.get(i));
1404     } else {
1405         dTotalError_kWh += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
1406         dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
1407         dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
1408
1409         if (dLineError > maxError) {
1410             maxError = dLineError;
1411         }
1412         if (dLineError < minError) {
1413             minError = dLineError;
1414         }
1415         errorFound = true;
1416     }
1417 } // end of kWh check
1418 } // end of LS
1419
1420 } // end of LS
1421
1422 if (errorFound == true) {
1423     errorsSize++;
1424     errorFound = false;
1425 }
1426
1427 }
1428
1429 } catch (Exception e) {
1430     System.out.println("Second: " + errorLine + "\tException:\t" + e);
1431 }
1432
1433 df = new DecimalFormat("#.###");
1434 df.setRoundingMode(RoundingMode.CEILING);
1435
1436 dTotalError = dTotalError * 5; // 10 7 5
1437 String label161 = "Total Error: " + df.format((dTotalError * 100) / formaliseMaxError);
1438 String label158 = "Max Error: " + df.format((maxError * 100) / formaliseMaxError) + "%";
1439 String label157 = "Min Error: " + df.format((minError * 100) / formaliseMaxError) + "%";

```

```

1442 // calculate Total Average Error
1443 double averageErrorTotal = toDouble(df.format((dTotalError / copiediotdata.size())));
1444 String label153 = "Total Average Error: " + df.format((averageErrorTotal * 100) / formaliseMaxError) + "%";
1445 totalaverageerror = toDouble(df.format((averageErrorTotal * 100) / formaliseMaxError));
1446
1447 // calculate Budget Average Error
1448 double averageError_Budget = toDouble(df.format((dTotalError_kWh / copiediotdata.size())));
1449 String label151 = "Budget Average Error: " + df.format((averageError_Budget * 100) / formaliseMaxError) + "%";
1450 budgeterror = toDouble(df.format((averageError_Budget * 100) / formaliseMaxError));
1451 // calculate Convenience Average Error
1452 double averageError_Convenience = toDouble(df.format(((dTotalError - dTotalError_kWh) / copiediotdata.size())));
1453 String label152 = "Convenience Average Error: "
1454     + df.format((averageError_Convenience * 100) / formaliseMaxError) + "%";
1455 convenienceerror = toDouble(df.format((averageError_Convenience * 100) / formaliseMaxError));
1456
1457 System.out.println(label161);
1458 System.out.println(label158);
1459 System.out.println(label157);
1460 System.out.println(label153);
1461 System.out.println(label151);
1462 System.out.println(label152);
1463
1464 copiediotdata3.clear();
1465
1466 // *****SECOND RUN*****
1467 Double meansum = 0.0;
1468 maxError = 0.0;
1469 minError = 100.0;
1470 dTotalError = 0;
1471 dLineError = 0;
1472
1473 try {
1474     for (int i = 0; i < copiediotdata.size(); i++) {
1475         dLineError = 0; // gets the total record line error in regards of all the ranges
1476         for (int k = 0; k < listRanges.length; k++) {
1477             errorLine = makeDataLine(copiediotdata.get(i));
1478
1479             String[] copiediotdataSplitted2 = copiediotdata.get(i);
1480             String[] rangesSplitted = listRanges[k];
1481
1482             if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
1483             {
1484                 // checks time and temperature and if season is empty
1485
1486                 if (((!rangesSplitted[0].equals("SUMMER")) || (!rangesSplitted[0].equals("WINTER"))))
1487                     && (rangesSplitted[3].equals("TEMPER."))) {
1488                     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1489                         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1490                         && toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
1491                         && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
1492                         copiediotdata3.add(copiediotdata.get(i));
1493                     } else {
1494                         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1495                         if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1496                             && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1497                             && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1498                             dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1499                             dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1500
1501                             if (dLineError > maxError) {
1502                                 maxError = dLineError;
1503                             }
1504                             if (dLineError < minError) {
1505                                 minError = dLineError;
1506                             }
1507                         } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1508                             && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1509                             && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1510                             dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1511                             dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1512
1513                             if (dLineError > maxError) {
1514                                 maxError = dLineError;
1515                             }
1516                             if (dLineError < minError) {
1517                                 minError = dLineError;
1518                             }
1519                         }
1520                     }
1521                 }
1522             }
1523         }
1524     }
1525
1526     if (((rangesSplitted[0].equals("SUMMER")) || (rangesSplitted[0].equals("WINTER"))))
1527         && (rangesSplitted[3].equals("TEMPER."))) {
1528         int iSeason = getMonth(copiediotdataSplitted2[0]);
1529
1530         // checks if SUMMER
1531         if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {

```



```

1532 if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
1533     && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
1534     copiediotdata3.add(copiediotdata.get(i));
1535 }
1536 } else {
1537     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1538     if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1539         dTotalError += toDouble(rangesSplitted[1])
1540             - toDouble(copiediotdataSplitted2[3]);
1541         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1542
1543         if (dLineError > maxError) {
1544             maxError = dLineError;
1545         }
1546         if (dLineError < minError) {
1547             minError = dLineError;
1548         }
1549     } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1550         dTotalError += toDouble(copiediotdataSplitted2[3])
1551             - toDouble(rangesSplitted[2]);
1552         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1553
1554         if (dLineError > maxError) {
1555             maxError = dLineError;
1556         }
1557         if (dLineError < minError) {
1558             minError = dLineError;
1559         }
1560     }
1561 }
1562 }
1563 // checks if WINTER
1564 else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
1565     if (toDouble(copiediotdataSplitted2[3]) >= toDouble(rangesSplitted[1])
1566         && toDouble(copiediotdataSplitted2[3]) <= toDouble(rangesSplitted[2])) {
1567         copiediotdata3.add(copiediotdata.get(i));
1568     }
1569 } else {
1570     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1571     if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1572         dTotalError += toDouble(rangesSplitted[1])
1573             - toDouble(copiediotdataSplitted2[3]);
1574         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1575
1576         if (dLineError > maxError) {
1577             maxError = dLineError;
1578         }
1579         if (dLineError < minError) {
1580             minError = dLineError;
1581         }
1582     } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1583         dTotalError += toDouble(copiediotdataSplitted2[3])
1584             - toDouble(rangesSplitted[2]);
1585         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1586
1587         if (dLineError > maxError) {
1588             maxError = dLineError;
1589         }
1590         if (dLineError < minError) {
1591             minError = dLineError;
1592         }
1593     }
1594 }
1595 }
1596 }
1597 }
1598 }
1599 if (rangesSplitted[0].equals("kWh")) {
1600     if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
1601         copiediotdata3.add(copiediotdata.get(i));
1602     } else {
1603         dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
1604         dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
1605         //
1606
1607         if (dLineError > maxError) {
1608             maxError = dLineError;
1609         }
1610         if (dLineError < minError) {
1611             minError = dLineError;
1612         }
1613         errorFound = true;
1614     }
1615 }
1616 }
1617 }
1618 if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
1619 {
1620     // checks LIGHT

```

```

1622 if (rangesSplitted[3].equals("LIGHT")) {
1623     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1624         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1625         && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1626         && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1627         copiediotdata3.add(copiediotdata.get(i));
1628     }
1629 } else {
1630     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1631     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1632         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1633         && toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1634         dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1635         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1636     }
1637     if (dLineError > maxError) {
1638         maxError = dLineError;
1639     }
1640     if (dLineError < minError) {
1641         minError = dLineError;
1642     }
1643 } else if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1644     && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1645     && toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1646     dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1647     dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1648 }
1649 if (dLineError > maxError) {
1650     maxError = dLineError;
1651 }
1652 if (dLineError < minError) {
1653     minError = dLineError;
1654 }
1655 }
1656 } // end of: checks LIGHT
1657
1658 // checks SUNNY
1659 if (rangesSplitted[3].equals("SUNNY")) {
1660     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1661         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1662         && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1663         && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1664         copiediotdata3.add(copiediotdata.get(i));
1665     }
1666 }
1667
1668 } else {
1669     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1670     if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1671         dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1672         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1673     }
1674     if (dLineError > maxError) {
1675         maxError = dLineError;
1676     }
1677     if (dLineError < minError) {
1678         minError = dLineError;
1679     }
1680 } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1681     dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1682     dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1683 }
1684 if (dLineError > maxError) {
1685     maxError = dLineError;
1686 }
1687 if (dLineError < minError) {
1688     minError = dLineError;
1689 }
1690 }
1691 } // end of :checks SUNNY
1692
1693 // checks CLOUDY
1694 if (rangesSplitted[3].equals("CLOUDY")) {
1695     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1696         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1697         && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1698         && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1699         copiediotdata3.add(copiediotdata.get(i));
1700     }
1701 } else {
1702     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1703     if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1704         dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1705         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1706     }
1707     if (dLineError > maxError) {
1708         maxError = dLineError;
1709     }
1710     if (dLineError < minError) {
1711         minError = dLineError;
1712     }
1713 }

```

```

1712     }
1713     } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1714         dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1715         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1716
1717         if (dLineError > maxError) {
1718             maxError = dLineError;
1719         }
1720         if (dLineError < minError) {
1721             minError = dLineError;
1722         }
1723     }
1724 } // end of :checks CLOUDY
1725
1726 // checks DOOR OPEN-LIGHT
1727 if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
1728     if ((getHour(copiediotdataSplitted2[0]) >= ruleHourFrom(rangesSplitted[0])
1729         && getHour(copiediotdataSplitted2[0]) <= ruleHourTo(rangesSplitted[0]))
1730         && toInt(copiediotdataSplitted2[3]) >= toInt(rangesSplitted[1])
1731         && toInt(copiediotdataSplitted2[3]) <= toInt(rangesSplitted[2])) {
1732         copiediotdata3.add(copiediotdata.get(i));
1733     }
1734 } else {
1735     // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1736     if (toDouble(copiediotdataSplitted2[3]) < toDouble(rangesSplitted[1])) {
1737         dTotalError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1738         dLineError += toDouble(rangesSplitted[1]) - toDouble(copiediotdataSplitted2[3]);
1739
1740         if (dLineError > maxError) {
1741             maxError = dLineError;
1742         }
1743         if (dLineError < minError) {
1744             minError = dLineError;
1745         }
1746     }
1747     } else if (toDouble(copiediotdataSplitted2[3]) > toDouble(rangesSplitted[2])) {
1748         dTotalError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1749         dLineError += toDouble(copiediotdataSplitted2[3]) - toDouble(rangesSplitted[2]);
1750
1751         if (dLineError > maxError) {
1752             maxError = dLineError;
1753         }
1754         if (dLineError < minError) {
1755             minError = dLineError;
1756         }
1757     }
1758 } // end of :checks DOOR OPEN-LIGHT
1759
1760 if (rangesSplitted[0].equals("kWh")) {
1761     if (toDouble(copiediotdataSplitted2[7]) <= toDouble(rangesSplitted[1])) {
1762         copiediotdata3.add(copiediotdata.get(i));
1763     }
1764 } else {
1765     dTotalError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
1766     dLineError += toDouble(copiediotdataSplitted2[7]) - toDouble(rangesSplitted[2]);
1767
1768     if (dLineError > maxError) {
1769         maxError = dLineError;
1770     }
1771     if (dLineError < minError) {
1772         minError = dLineError;
1773     }
1774     errorFound = true;
1775 }
1776 } // end of kWh check
1777
1778 } // end of LS
1779
1780 }
1781
1782 meansSum = meansSum + Math.pow(dLineError - averageErrorTotal, 2);
1783
1784 }
1785 } catch (Exception e) {
1786     System.out.println("Second: " + errorLine + "\tException:\t" + e);
1787 }
1788
1789 df = new DecimalFormat("#.####");
1790 df.setRoundingMode(RoundingMode.CEILING);
1791
1792 double standardDeviation = Math.sqrt(meansSum / copiediotdata.size());
1793 String label156 = "Standard Deviation: " + df.format((standardDeviation * 100) / formaliseMaxError) + "%";
1794 deviation = toDouble(df.format((standardDeviation * 100) / formaliseMaxError));
1795 System.out.println(label156);
1796 copiediotdata3.clear();
1797
1798 postData();
1799 } // END OF ENERGY PLANNER - number of modifications
1800

```

GreenPlanner.java

```
1  /*
2      This file implements the Green Planner algorithm.
3      Copyright (C) 2021 Antonis Vasileiou
4      This program is free software: you can redistribute it and/or modify
5      it under the terms of the GNU General Public License as published by
6      the Free Software Foundation, either version 3 of the License, or
7      (at your option) any later version.
8      This program is distributed in the hope that it will be useful,
9      but WITHOUT ANY WARRANTY; without even the implied warranty of
10     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
11     GNU General Public License for more details.
12     You should have received a copy of the GNU General Public License
13     along with this program. If not, see https://www.gnu.org/licenses/gpl-3.0.html.
14 */
15 package iotprojectpackage;
16
17 import java.io.BufferedReader;
18 import java.io.DataOutputStream;
19 import java.io.FileWriter;
20 import java.io.InputStreamReader;
21 import java.math.RoundingMode;
22 import java.net.HttpURLConnection;
23 import java.net.URL;
24 import java.nio.charset.StandardCharsets;
25 import java.text.DecimalFormat;
26 import java.text.ParseException;
27 import java.text.SimpleDateFormat;
28 import java.time.ZoneId;
29 import java.util.ArrayList;
30 import java.util.Calendar;
31 import java.util.Date;
32 import java.util.GregorianCalendar;
33 import java.util.Random;
34
35 import com.google.gson.Gson;
36 import com.google.gson.JsonArray;
37 import com.google.gson.JsonObject;
38 import com.google.gson.JsonParser;
39
40 public class GreenPlanner {
41
42     private String errorLine; // global
43
44     private SimpleDateFormat simpleFormatter = new SimpleDateFormat("yyyy-MM-dd HH:mm:ss");
45     private SimpleDateFormat getHour = new SimpleDateFormat("HH");
46     private Date dtTemp;
47
48     private double contiguous_ms = 0;
49     private String monthlyKWH_listAssignment = "original"; // to help identify allowed kwh ArrayList->original,
49                                     // 1%, 5%, 10% or 20%
50     private ArrayList<String[]> copiediotdata;
51     private ArrayList<String[]> GreenPlannerData;
52     private ArrayList<String[]> copiediotdata3;
53     private ArrayList<String[]> MRTrules;
54     private boolean solutionsFound;
55     private ArrayList<String[]> recursionResults = new ArrayList<String[]>();
56     private double public ms = 0;
57     private int recursiveIterator = 0;
58     private String urlString = "http://10.16.30.73/imcfqp/api/update-results";
59     private double totalaverageerror = 0.0;
60     private double budgeterror = 0.0;
61     private double convinienceerror = 0.0;
62     private double deviation = 0.0;
63     private double totalkwh = 0.0;
64     private double total_daily_ms_CO2 = 0;
65     private double kwh_with_CO2 = 0;
66     private double kwh_without_CO2 = 0;
67     private double CO2_kg_emissions = 0;
68
69     private String listRanges[];
70     private String kwhpermonth = "";
71
72     public GreenPlanner() {
73         solutionsFound = false;
74     }
75
76     public void ArrayListFromArray_toStringArray(ArrayList<String> ArrayList_from,
77         ArrayList<String[]> ArrayList_to) {
78
79         for (int i = 0; i < ArrayList_from.size(); i++) {
80             String temp[] = ArrayList_from.get(i).split("\\|");
81             ArrayList_to.add(temp);
82         }
83     }
84
85     public String makeDataLine(String[] array) {
86         String s = "";
87         String seperator = "|";
88         int count = 0;
89         int length = array.length;
90     }
```

```

92     for (String temp : array) {
93         if (count == length - 1) {
94             s = s.concat(temp);
95         } else {
96             s = s.concat(temp);
97             s = s.concat(seperator);
98         }
99         count++;
100     }
101
102     return s;
103 }
104
105 public static String[] addX(String myarray[], String ele) {
106
107     int n = myarray.length;
108     String newArray[] = new String[n + 1];
109     // copy original array into new array
110     for (int i = 0; i < n; i++)
111         newArray[i] = myarray[i];
112
113     // add element to the new array
114     newArray[n] = ele;
115
116     return newArray;
117 }
118
119 // converts string to double
120 public double toDouble(String str) {
121
122     return Double.parseDouble(str);
123 }
124
125 // converts string to double
126 public int toInt(String str) {
127
128     return Integer.parseInt(str);
129 }
130
131
132
133 // returns hour from record datetime
134 public int getHour(String date) {
135
136     return toInt(date.substring(11, 13));
137 }
138
139
140
141 // returns month from record datetime
142 public int getMonth(String date) {
143
144     return toInt(date.substring(5, 7));
145 }
146
147
148 // returns month from record datetime
149 public int getDay(String date) {
150
151     return toInt(date.substring(8, 10));
152 }
153
154
155 // returns the hour the rule starts
156 public int ruleHourFrom(String ruletime) {
157
158     return toInt(ruletime.substring(0, 2));
159 }
160
161
162 // returns the hour the rule ends
163 public int ruleHourTo(String ruletime) {
164
165     return toInt(ruletime.substring(6, 8));
166 }
167
168
169
170
171 public void postData() {
172     try {
173
174         String urlParameters = "gp_kWh=" + totalkWh + "&gp_totalError=" + totalaverageerror
175             + "&gp_convenienceError=" + convenienceerror + "&gp_budgetError=" + budgeterror
176             + "&gp_standardDeviation=" + deviation + "&gp_kWh_per_month=" + kWhpermonth + "&gp_kWh_with_co2="
177             + kWh_with_CO2 + "&gp_kWh_without_co2=" + kWh_without_CO2 + "&gp_co2_emissions_kg="
178             + CO2_kg_emissions;
179         byte[] postData = urlParameters.getBytes(StandardCharsets.UTF_8);
180         int postDataLength = postData.length;
181

```

```

182     URL url = new URL(urlString);
183     HttpURLConnection conn = (HttpURLConnection) url.openConnection();
184
185     conn.setDoOutput(true);
186     conn.setRequestMethod("POST");
187     conn.setRequestProperty("Content-Type", "application/x-www-form-urlencoded");
188     conn.setRequestProperty("Charset", "UTF-8");
189     conn.setRequestProperty("Content-Length", Integer.toString(postDataLength));
190     conn.setUseCaches(false);
191
192     conn.connect();
193
194     DataOutputStream wr = new DataOutputStream(conn.getOutputStream());
195     wr.write(postData);
196     conn.getResponseCode();
197
198 } catch (Exception e) {
199     for (StackTraceElement s : e.getStackTrace())
200         System.out.println(s);
201 }
202
203
204 // START OF CALCULATE KWH METHOD - number of modifications (GREEN PLANNER)
205 public double calculateKWH5(ArrayList<String[]> sRecordsDaily) {
206
207     ArrayList<String[]> copiediotdatakwh = new ArrayList<String[]>(sRecordsDaily);
208     ArrayList<Integer> removedMetaRules = new ArrayList<Integer>();
209
210     Random rand = new Random();
211     boolean startALLzeros = false;
212     boolean startALLrandom = false;
213     int numofloops = 3;
214     boolean randomRule[] = new boolean[listRanges.length];
215     int numrules = listRanges.length;
216     int random;
217     if (removedMetaRules.size() == 0) {
218         if (startALLzeros == true) {
219             for (int d = 0; d < randomRule.length; d++) {
220                 randomRule[d] = false;
221                 removedMetaRules.add(d);
222             }
223         } else if (startALLrandom == true) {
224             for (int d = 0; d < randomRule.length; d++) {
225                 // generates random number from 0 to 1
226                 random = rand.nextInt(2);
227
228                 if (random == 0) {
229                     randomRule[d] = false;
230                     removedMetaRules.add(d);
231                 } else {
232                     randomRule[d] = true;
233                 }
234             }
235         }
236     } else {
237         for (int d = 0; d < randomRule.length; d++) {
238             randomRule[d] = true;
239         }
240     }
241
242     // for predefined times add or remove rules
243     for (int i = 0; i < numofloops; i++) {
244         random = rand.nextInt(numrules);
245
246         if (randomRule[random]) {
247             randomRule[random] = false;
248             removedMetaRules.add(random);
249         } else {
250             randomRule[random] = true;
251             removedMetaRules.remove(Integer.valueOf(random));
252         }
253     }
254
255 }
256
257 if (solutionsFound == false) {
258
259     for (int i = 0; i < numofloops; i++) {
260         random = rand.nextInt(numrules);
261
262         if (randomRule[random]) {
263             randomRule[random] = false;
264             removedMetaRules.add(random);
265         } else {
266             randomRule[random] = true;
267             removedMetaRules.remove(Integer.valueOf(random));
268         }
269     }
270
271 } else if (solutionsFound == true) {

```



```

272     for (int d = 0; d < randomRule.length; d++) {
273         randomRule[d] = true;
274     }
275
276     for (int i = 0; i < removedMetaRules.size(); i++) {
277         randomRule[removedMetaRules.get(i)] = false;
278     }
279
280 }
281
282 int IFTTTconstraintsApplied = 0;
283 double milliWatts = 0;
284 Date dt1 = dtTemp;
285 Date dt2;
286 double span;
287 double ms = 0;
288 double localContiguous_ms = contiguous_ms;
289 double daily_ms_CO2 = 0;
290 ArrayList<Double> monthlyKWH = new ArrayList<Double>();
291 Calendar cdt1 = GregorianCalendar.getInstance(); // creates a new calendar instance
292 Calendar cdt2 = GregorianCalendar.getInstance();
293
294
295 if (monthlyKWH_listAssignment.equals("original")) {
296     monthlyKWH = new ArrayList<Double>();
297     // [] d= new [] { 25.02, 17.06, 7.96, 4.55, 5.69, 6.82, 7.96, 10.23,
298     // 6.82, 5.69, 6.82, 13.65 };
299     double[] d = new double[] { 1091.81, 744.42, 347.39, 198.51, 248.14, 297.77, 347.39, 446.65, 297.77, 248.14, 297.77, 595.53 };
300     for (int i = 0; i < d.length; i++)
301         monthlyKWH.add(d[i]);
302 } else if (monthlyKWH_listAssignment.equals("1")) {
303     monthlyKWH = new ArrayList<>();
304     double[] d = new double[] { 25.02, 17.06, 7.96, 4.55, 5.69, 6.82, 7.96, 10.23, 6.82, 5.69, 6.82, 13.65 };
305     for (int i = 0; i < d.length; i++)
306         monthlyKWH.add(d[i]);
307 } else if (monthlyKWH_listAssignment.equals("5")) {
308     monthlyKWH = new ArrayList<>();
309
310     double[] d = new double[] { 23.77, 16.21, 7.56, 4.32, 5.40, 6.48, 7.56, 9.72, 6.48, 5.40, 6.48, 12.96 };
311     for (int i = 0; i < d.length; i++)
312         monthlyKWH.add(d[i]);
313 } else if (monthlyKWH_listAssignment.equals("10")) {
314     monthlyKWH = new ArrayList<>();
315
316     double[] d = new double[] { 22.52, 15.35, 7.17, 4.09, 5.12, 6.14, 7.17, 9.21, 6.14, 5.12, 6.14, 12.28 };
317
318     for (int i = 0; i < d.length; i++)
319         monthlyKWH.add(d[i]);
320 } else if (monthlyKWH_listAssignment.equals("20")) {
321     monthlyKWH = new ArrayList<>();
322
323     double[] d = new double[] { 20.01, 13.65, 6.37, 3.64, 4.55, 5.46, 6.37, 8.19, 5.46, 4.55, 5.46, 10.92 };
324     for (int i = 0; i < d.length; i++)
325         monthlyKWH.add(d[i]);
326 } else if (monthlyKWH_listAssignment.equals("30")) {
327     monthlyKWH = new ArrayList<>();
328     double[] d = new double[] { 17.51, 11.94, 5.57, 3.18, 3.98, 4.78, 5.57, 7.16, 4.78, 3.98, 4.78, 9.55 };
329     for (int i = 0; i < d.length; i++)
330         monthlyKWH.add(d[i]);
331 } else if (monthlyKWH_listAssignment.equals("40")) {
332     monthlyKWH = new ArrayList<>();
333     double[] d = new double[] { 15.01, 10.24, 4.78, 2.73, 3.41, 4.09, 4.78, 6.14, 4.09, 3.41, 4.09, 8.19 };
334     for (int i = 0; i < d.length; i++)
335         monthlyKWH.add(d[i]);
336 }
337
338 try {
339     ArrayList<String> listRules = new ArrayList<String>();
340
341     for (int d = 0; d < randomRule.length; d++) {
342         if (randomRule[d]) {
343             // temp
344             if (listRanges[d][3].equals("TEMPER.")) {
345                 listRules.add(listRanges[d][0] + "|" + "+" + listRanges[d][1] + "|" + "---" + "|"
346                     + listRanges[d][1] + "|" + "---" + "|" + "---" + "|" + "---" + "|" + "SET TEMPERATURE");
347             }
348             // light
349             if (listRanges[d][3].equals("LIGHT")) {
350                 listRules.add(listRanges[d][0] + "|" + "+" + listRanges[d][1] + "|"
351                     + listRanges[d][1] + "|" + "---" + "|" + "---" + "|" + "---" + "|" + "SET LIGHT");
352             }
353             IFTTTconstraintsApplied++;
354         }
355     }
356 }
357
358 for (int i = 0; i < copiediotdatakwh.size(); i++) {
359     for (int k = 0; k < listRules.size(); k++) {
360         errorLine = String.join("|", copiediotdatakwh.get(i));
361     }

```

```

362
363 String[] copiediotdataSplitted = copiediotdatakwh.get(i);
364 String[] rulesSplitted = listRules.get(k).split("\\|");
365
366 if (copiediotdataSplitted[2].startsWith("T")) // gets T for temperatures
367 {
368     // checks time and temperature and if season,wether,door is empty
369     if (!rulesSplitted[1].equals("----") && rulesSplitted[4].equals("----")
370         && rulesSplitted[5].equals("----") && rulesSplitted[6].equals("----")) {
371
372         if (getHour(copiediotdataSplitted[0]) >=
373             ruleHourFrom(rulesSplitted[0])
374             && getHour(copiediotdataSplitted[0]) <=
375                 ruleHourTo(rulesSplitted[0])) {
376             dt2 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
377             span = Math.abs(dt2.getTime() - dt1.getTime());
378             ms += span;
379             dt1 = dt2;
380             cdt2 = GregorianCalendar.getInstance(); // creates a new calendar instance
381             cdt2.setTime(dt2); // assigns calendar to given date
382
383             if (cdt2.get(Calendar.HOUR) >= 8 && cdt2.get(Calendar.HOUR) <= 15
384                 && copiediotdataSplitted[4].equals("sky is clear")) {
385                 daily_ms_CO2 += span;
386             }
387
388             localContiguous_ms += span;
389             double secondsTemp = localContiguous_ms / 1000;
390             double hoursTemp = secondsTemp / 3600;
391             double kWhTemp = ((hoursTemp * 320) / 1000) * 1100;
392             copiediotdataSplitted = addX(copiediotdataSplitted, String.valueOf(kWhTemp));
393
394             milliWatts = milliWatts + 320;
395             copiediotdataSplitted[3] = rulesSplitted[3];
396             copiediotdatakwh.set(i, copiediotdataSplitted);
397             break; // its already on the ArrayList no need to add it twice
398         }
399     }
400 }
401
402
403
404
405 if (copiediotdataSplitted[2].startsWith("LS")) // gets LS for light sensor
406 {
407     // checks LIGHT , and if weather,door is empty
408     if (!rulesSplitted[2].equals("----") && rulesSplitted[5].equals("----")
409         && rulesSplitted[6].equals("----")) {
410         // checks time and light value
411         if (getHour(copiediotdataSplitted[0]) >=
412             ruleHourFrom(rulesSplitted[0])
413             && getHour(copiediotdataSplitted[0]) <=
414                 ruleHourTo(rulesSplitted[0])) {
415             dt2 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
416             span = Math.abs(dt2.getTime() - dt1.getTime());
417             ms += span;
418             dt1 = dt2;
419             cdt2 = GregorianCalendar.getInstance(); // creates a new calendar instance
420             cdt2.setTime(dt2); // assigns calendar to given date
421             if (cdt2.get(Calendar.HOUR) >= 8 && cdt2.get(Calendar.HOUR) <= 15
422                 && copiediotdataSplitted[4].equals("sky is clear")) {
423                 daily_ms_CO2 += span;
424             }
425
426             localContiguous_ms += span;
427             double secondsTemp = localContiguous_ms / 1000;
428             double hoursTemp = secondsTemp / 3600;
429             double kWhTemp = ((hoursTemp * 320) / 1000) * 1100;
430             copiediotdataSplitted = addX(copiediotdataSplitted, String.valueOf(kWhTemp));
431
432             copiediotdataSplitted[3] = rulesSplitted[3];
433             // copiediotdata2.add(String.Join("|", copiediotdataSplitted));
434             copiediotdatakwh.set(i, copiediotdataSplitted);
435             break;
436         }
437     } // end of : //checks LIGHT and if weather is empty
438 }
439
440
441
442 if (k == listRules.size() - 1) {
443     dt1 = simpleFormatter.parse(copiediotdataSplitted[0].substring(0, 18));
444 }
445
446 }
447
448 String[] copiediotdataSplittedAfterIFTTT = copiediotdatakwh.get(i);
449 if (copiediotdataSplittedAfterIFTTT.length < 8) {
450     copiediotdatakwh.set(i, addX(copiediotdataSplittedAfterIFTTT, "00"));
451 }

```



```

452     }
453 }
454 } catch (Exception ex) {
455     System.out.println("First B: " + errorLine + " => Exception: " + ex);
456 }
457
458 // calculations for TIME(hours), watts and kWh
459 double seconds = ms / 1000;
460 double hours = seconds / 3600;
461
462 double kWh = ((hours * 320) / 1000) * 1100;
463
464 double seconds_CO2 = daily_ms_CO2 / 1000;
465 double hours_CO2 = seconds_CO2 / 3600;
466 double kWh_CO2 = ((hours_CO2 * 320) / 1000) * 1100;
467
468 String[] copiediotdataSplitted_Month = copiediotdatakwh.get(0);
469 Date Month;
470 Calendar calendar;
471 int currentMonth = 0;
472 try {
473     Month = simpleFormatter.parse(copiediotdataSplitted_Month[0].substring(0, 18));
474     calendar = GregorianCalendar.getInstance(); // creates a new calendar instance
475     calendar.setTime(Month); // assigns calendar to given date
476     currentMonth = calendar.get(Calendar.MONTH);
477 } catch (ParseException e1) {
478     e1.printStackTrace();
479 }
480 // months from calendar are indexed from 0 therefore currentMonth=0 is January
481 double allowedMonthly_kWh = monthlyKWH.get(currentMonth) * 6.5; // 6.5, *7.9;
482
483 // *****SECOND AND LAST
484 // PHASE*****//
485 double dTotalError = 0;
486 double dLineError = 0;
487 double dTotalError_kWh = 0;
488 double maxError = 0;
489 double minError = 100;
490 boolean errorFound = false;
491 int errorsize = 0;
492
493 try {
494
495     for (int i = 0; i < copiediotdatakwh.size(); i++) {
496
497         dLineError = 0; // gets the total record line error in regards of all the ranges
498         for (int k = 0; k < listRanges.length; k++) {
499             errorLine = String.join("|", copiediotdatakwh.get(i));
500
501             String[] copiediotdataSplitted2 = copiediotdatakwh.get(i);
502             String[] rangesSplitted = listRanges[k];
503
504             if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
505             {
506
507                 // checks time and temperature and if season is empty
508                 if (!rangesSplitted[0].equals("SUMMER") || !rangesSplitted[0].equals("WINTER"))
509                     && rangesSplitted[3].equals("TEMPER.") {
510                     if ((getHour(copiediotdataSplitted2[0]) >=
511                         ruleHourFrom(rangesSplitted[0]))
512                         && getHour(copiediotdataSplitted2[0]) <=
513                             ruleHourTo(rangesSplitted[0])
514                         && toDouble(copiediotdataSplitted2[3]) >=
515                             toDouble(rangesSplitted[1])
516                         && toDouble(copiediotdataSplitted2[3]) <=
517                             toDouble(rangesSplitted[2])) {
518                         copiediotdata3.add(copiediotdatakwh.get(i));
519                         // its already on the ArrayList no need to add it twice
520                     } else {
521                         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
522                         if ((getHour(copiediotdataSplitted2[0]) >=
523                             ruleHourFrom(rangesSplitted[0]))
524                             && getHour(copiediotdataSplitted2[0]) <=
525                                 ruleHourTo(rangesSplitted[0])
526                             && toDouble(copiediotdataSplitted2[3]) <
527                                 toDouble(rangesSplitted[1])) {
528                             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
529                                 - toDouble(copiediotdataSplitted2[3])) / 1);
530                             dLineError = dLineError + ((toDouble(rangesSplitted[1])
531                                 - toDouble(copiediotdataSplitted2[3])) / 1);
532
533                             if (dLineError > maxError) {
534                                 maxError = dLineError;
535                             }
536                             if (dLineError < minError) {
537                                 minError = dLineError;
538                             }
539                             errorFound = true;
540                         } else if (getHour(copiediotdataSplitted2[0]) >=
541                             ruleHourFrom(rangesSplitted[0])

```

```

542         && getHour(copiediotdataSplitted2[0]) <=
543             ruleHourTo(rangesSplitted[0])
544         && toDouble(copiediotdataSplitted2[3]) >
545             toDouble(rangesSplitted[2]) ) {
546     dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
547         - toDouble(rangesSplitted[2])) / 1);
548     dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
549         - toDouble(rangesSplitted[2])) / 1);
550
551     if (dLineError > maxError) {
552         maxError = dLineError;
553     }
554     if (dLineError < minError) {
555         minError = dLineError;
556     }
557     errorFound = true;
558 }
559
560 }
561
562
563 if ((rangesSplitted[0].equals("SUMMER") || rangesSplitted[0].equals("WINTER"))
564     && rangesSplitted[3].equals("TEMPER.")) {
565     int iSeason = getMonth(copiediotdataSplitted2[0]);
566
567     // checks if SUMMER
568     if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
569         if (toDouble(copiediotdataSplitted2[3]) >=
570             toDouble(rangesSplitted[1])
571             && toDouble(copiediotdataSplitted2[3]) <=
572                 toDouble(rangesSplitted[2])) {
573             copiediotdata3.add(copiediotdatakwh.get(i));
574             // its already on the ArrayList no need to add it twice
575         } else {
576             // assigns TOTAL ERROR AND CURRENT RECORD ERROR
577             if (toDouble(copiediotdataSplitted2[3]) <
578                 toDouble(rangesSplitted[1])) {
579                 dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
580                     - toDouble(copiediotdataSplitted2[3])) / 1);
581                 dLineError = dLineError + ((toDouble(rangesSplitted[1])
582                     - toDouble(copiediotdataSplitted2[3])) / 1);
583
584                 if (dLineError > maxError) {
585                     maxError = dLineError;
586                 }
587
588                 if (dLineError < minError) {
589                     minError = dLineError;
590                 }
591                 errorFound = true;
592             } else if (toDouble(copiediotdataSplitted2[3]) >
593                 toDouble(rangesSplitted[2])) {
594                 dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
595                     - toDouble(rangesSplitted[2])) / 1);
596                 dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
597                     - toDouble(rangesSplitted[2])) / 1);
598
599                 if (dLineError > maxError) {
600                     maxError = dLineError;
601                 }
602                 if (dLineError < minError) {
603                     minError = dLineError;
604                 }
605                 errorFound = true;
606             }
607         }
608     }
609     // checks if WINTER
610     else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
611         if (toDouble(copiediotdataSplitted2[3]) >=
612             toDouble(rangesSplitted[1])
613             && toDouble(copiediotdataSplitted2[3]) <=
614                 toDouble(rangesSplitted[2])) {
615             copiediotdata3.add(copiediotdatakwh.get(i));
616             // its already on the ArrayList no need to add it twice
617         } else {
618             // assigns TOTAL ERROR AND CURRENT RECORD ERROR
619             if (toDouble(copiediotdataSplitted2[3]) <
620                 toDouble(rangesSplitted[1])) {
621                 dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
622                     - toDouble(copiediotdataSplitted2[3])) / 1);
623                 dLineError = dLineError + ((toDouble(rangesSplitted[1])
624                     - toDouble(copiediotdataSplitted2[3])) / 1);
625
626                 if (dLineError > maxError) {
627                     maxError = dLineError;
628                 }
629                 if (dLineError < minError) {
630                     minError = dLineError;
631                 }
632                 errorFound = true;

```

```

632 } else if (toDouble(copiediotdataSplitted2[3]) >
633     toDouble(rangesSplitted[2])) {
634     dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
635         - toDouble(rangesSplitted[2])) / 1);
636     dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
637         - toDouble(rangesSplitted[2])) / 1);
638
639     if (dLineError > maxError) {
640         maxError = dLineError;
641     }
642     if (dLineError < minError) {
643         minError = dLineError;
644     }
645     errorFound = true;
646 }
647 }
648 }
649
650 }
651
652 if (rangesSplitted[0].equals("kWh")) {
653     if (toDouble(copiediotdataSplitted2[7]) <=
654         toDouble(rangesSplitted[1])) {
655         copiediotdata3.add(copiediotdatakwh.get(i));
656     } else {
657         dTotalError_kWh = dTotalError_kWh + ((toDouble(copiediotdataSplitted2[7])
658             - toDouble(rangesSplitted[2])) / 1);
659         dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
660             - toDouble(rangesSplitted[2])) / 1);
661         dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
662             - toDouble(rangesSplitted[2])) / 1);
663
664         if (dLineError > maxError) {
665             maxError = dLineError;
666         }
667         if (dLineError < minError) {
668             minError = dLineError;
669         }
670         errorFound = true;
671     }
672 }
673
674 }
675
676 if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor

```

```

677 {
678     // checks LIGHT
679     if (rangesSplitted[3].equals("LIGHT")) {
680         if (getHour(copiediotdataSplitted2[0]) >=
681             ruleHourFrom(rangesSplitted[0])
682             && getHour(copiediotdataSplitted2[0]) <=
683                 ruleHourTo(rangesSplitted[0])
684             && toInt(copiediotdataSplitted2[3]) >=
685                 toInt(rangesSplitted[1])
686             && toInt(copiediotdataSplitted2[3]) <=
687                 toInt(rangesSplitted[2])) {
688             copiediotdata3.add(copiediotdatakwh.get(i));
689             // its already on the ArrayList no need to add it twice
690         }
691         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
692         if (getHour(copiediotdataSplitted2[0]) >=
693             ruleHourFrom(rangesSplitted[0])
694             && getHour(copiediotdataSplitted2[0]) <=
695                 ruleHourTo(rangesSplitted[0])
696             && toDouble(copiediotdataSplitted2[3]) <
697                 toDouble(rangesSplitted[1])) {
698             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
699                 - toDouble(copiediotdataSplitted2[3])) / 1);
700             dLineError = dLineError + ((toDouble(rangesSplitted[1])
701                 - toDouble(copiediotdataSplitted2[3])) / 1);
702
703             if (dLineError > maxError) {
704                 maxError = dLineError;
705             }
706             if (dLineError < minError) {
707                 minError = dLineError;
708             }
709             errorFound = true;
710         } else if (getHour(copiediotdataSplitted2[0]) >=
711             ruleHourFrom(rangesSplitted[0])
712             && getHour(copiediotdataSplitted2[0]) <=
713                 ruleHourTo(rangesSplitted[0])
714             && toDouble(copiediotdataSplitted2[3]) >
715                 toDouble(rangesSplitted[2])) {
716             dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
717                 - toDouble(rangesSplitted[2])) / 1);
718             dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
719                 - toDouble(rangesSplitted[2])) / 1);
720
721             if (dLineError > maxError) {

```

```

722         maxError = dLineError;
723     }
724     if (dLineError < minError) {
725         minError = dLineError;
726     }
727     errorFound = true;
728 }
729 }
730 } // end of: checks LIGHT
731
732 // checks SUNNY
733 if (rangesSplitted[3].equals("SUNNY")) {
734     if (getHour(copiediotdataSplitted2[0]) >=
735         ruleHourFrom(rangesSplitted[0])
736         && getHour(copiediotdataSplitted2[0]) <=
737             ruleHourTo(rangesSplitted[0])
738         && toInt(copiediotdataSplitted2[3]) >=
739             toInt(rangesSplitted[1])
740         && toInt(copiediotdataSplitted2[3]) <=
741             toInt(rangesSplitted[2])) {
742         copiediotdata3.add(copiediotdatakw.get(i));
743         // its already on the ArrayList no need to add it twice
744     } else {
745         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
746         if (toDouble(copiediotdataSplitted2[3]) <
747             toDouble(rangesSplitted[1])) {
748             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
749                 - toDouble(copiediotdataSplitted2[3])) / 1);
750             dLineError = dLineError + ((toDouble(rangesSplitted[1])
751                 - toDouble(copiediotdataSplitted2[3])) / 1);
752
753             if (dLineError > maxError) {
754                 maxError = dLineError;
755             }
756             if (dLineError < minError) {
757                 minError = dLineError;
758             }
759             errorFound = true;
760         } else if (toDouble(copiediotdataSplitted2[3]) >
761             toDouble(rangesSplitted[2])) {
762             dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
763                 - toDouble(rangesSplitted[2])) / 1);
764             dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
765                 - toDouble(rangesSplitted[2])) / 1);
766
767             if (dLineError > maxError) {
768                 maxError = dLineError;
769             }
770             if (dLineError < minError) {
771                 minError = dLineError;
772             }
773             errorFound = true;
774         }
775     }
776 } // end of :checks SUNNY
777
778 // checks CLOUDY
779 if (rangesSplitted[3].equals("CLOUDY")) {
780     if (getHour(copiediotdataSplitted2[0]) >=
781         ruleHourFrom(rangesSplitted[0])
782         && getHour(copiediotdataSplitted2[0]) <=
783             ruleHourTo(rangesSplitted[0])
784         && toInt(copiediotdataSplitted2[3]) >=
785             toInt(rangesSplitted[1])
786         && toInt(copiediotdataSplitted2[3]) <=
787             toInt(rangesSplitted[2])) {
788         copiediotdata3.add(copiediotdatakw.get(i));
789         // its already on the ArrayList no need to add it twice
790     } else {
791         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
792         if (toDouble(copiediotdataSplitted2[3]) <
793             toDouble(rangesSplitted[1])) {
794             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
795                 - toDouble(copiediotdataSplitted2[3])) / 1);
796             dLineError = dLineError + ((toDouble(rangesSplitted[1])
797                 - toDouble(copiediotdataSplitted2[3])) / 1);
798
799             if (dLineError > maxError) {
800                 maxError = dLineError;
801             }
802             if (dLineError < minError) {
803                 minError = dLineError;
804             }
805             errorFound = true;
806         } else if (toDouble(copiediotdataSplitted2[3]) >
807             toDouble(rangesSplitted[2])) {
808             dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
809                 - toDouble(rangesSplitted[2])) / 1);
810             dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
811                 - toDouble(rangesSplitted[2])) / 1);

```

```

812         if (dLineError > maxError) {
813             maxError = dLineError;
814         }
815         if (dLineError < minError) {
816             minError = dLineError;
817         }
818         errorFound = true;
819     }
820 }
821 } // end of :checks CLOUDY
822
823 // checks DOOR OPEN-LIGHT
824 if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
825     if (getHour(copiediotdataSplitted2[0]) >=
826         ruleHourFrom(rangesSplitted[0])
827         && getHour(copiediotdataSplitted2[0]) <=
828             ruleHourTo(rangesSplitted[0])
829         && toInt(copiediotdataSplitted2[3]) >=
830             toInt(rangesSplitted[1])
831         && toInt(copiediotdataSplitted2[3]) <=
832             toInt(rangesSplitted[2])) {
833         copiediotdata3.add(copiediotdatakwh.get(i));
834         // its already on the ArrayList no need to add it twice
835     } else {
836         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
837         if (toDouble(copiediotdataSplitted2[3]) <
838             toDouble(rangesSplitted[1])) {
839             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
840                 - toDouble(copiediotdataSplitted2[3])) / 1);
841             dLineError = dLineError + ((toDouble(rangesSplitted[1])
842                 - toDouble(copiediotdataSplitted2[3])) / 1);
843
844             if (dLineError > maxError) {
845                 maxError = dLineError;
846             }
847             if (dLineError < minError) {
848                 minError = dLineError;
849             }
850             errorFound = true;
851         } else if (toDouble(copiediotdataSplitted2[3]) >
852             toDouble(rangesSplitted[2])) {
853             dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
854                 - toDouble(rangesSplitted[2])) / 1);
855             dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
856                 - toDouble(rangesSplitted[2])) / 1);
857
858             if (dLineError > maxError) {
859                 maxError = dLineError;
860             }
861             if (dLineError < minError) {
862                 minError = dLineError;
863             }
864             errorFound = true;
865         }
866     }
867 } // end of :checks DOOR OPEN-LIGHT
868
869 if (rangesSplitted[0].equals("kWh")) {
870     if (toDouble(copiediotdataSplitted2[7]) <=
871         toDouble(rangesSplitted[1])) {
872         copiediotdata3.add(copiediotdatakwh.get(i));
873     } else {
874         dTotalError_kWh = dTotalError_kWh + ((toDouble(copiediotdataSplitted2[7])
875             - toDouble(rangesSplitted[2])) / 1);
876         dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
877             - toDouble(rangesSplitted[2])) / 1);
878         dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
879             - toDouble(rangesSplitted[2])) / 1);
880
881         if (dLineError > maxError) {
882             maxError = dLineError;
883         }
884         if (dLineError < minError) {
885             minError = dLineError;
886         }
887         errorFound = true;
888     }
889 } // end of kWh check
890
891 } // end of LS
892
893 }
894
895 if (errorFound == true) {
896     errorsize++;
897     errorFound = false;
898 }
899
900 }
901

```

```

902 } catch (Exception e) {
903     System.out.println("Second B: " + errorLine);
904 }
905
906
907 copiediotdata3.clear();
908
909 // *****end of second phase (calculating Error
910 // phase)*****
911
912 if ((kWh > allowedMonthly_kWh) && (solutionsFound == false))// || (recursiveIterator<6)
913 {
914
915     if (removedMetaRules.size() == numrules) {
916         removedMetaRules.clear();
917     }
918
919     calculateKWH5(sRecordsDaily);
920     return public_ms;
921
922 } else if (recursiveIterator < 15) {
923     if (recursionResults.size() == 0)// check if result found first time
924     {
925         String temp[] = new String[4];
926         String t = "";
927         boolean first = true;
928         for (int d = 0; d < removedMetaRules.size(); d++) {
929             if (first) {
930                 t = removedMetaRules.get(d).toString();
931                 first = false;
932             } else {
933                 t = t.concat(", " + removedMetaRules.get(d));
934             }
935         }
936         temp[0] = t;
937         temp[1] = String.valueOf(kWh);
938         temp[2] = String.valueOf(dTotalError);
939         temp[3] = String.valueOf(kWh_CO2);
940         recursionResults.add(temp);
941     } else // now check if new result is better than the previous and replace it
942     {
943         String[] splitted = recursionResults.get(0);
944         int recIterMinusOne = recursiveIterator - 1;
945         double anneallingValue = recIterMinusOne - recIterMinusOne * (2 / 15);
946         Random randAnnealing = new Random();
947         int numRandAnnealing = randAnnealing.nextInt(100);

```

```

947
948         if (kWh_CO2 >= toDouble(splitted[3])) {
949
950             String temp[] = new String[4];
951             String t = "";
952             boolean first = true;
953             for (int d = 0; d < removedMetaRules.size(); d++) {
954                 if (first) {
955                     t = removedMetaRules.get(d).toString();
956                     first = false;
957                 } else {
958                     t = t.concat(", " + removedMetaRules.get(d));
959                 }
960             }
961             temp[0] = t;
962             temp[1] = String.valueOf(kWh);
963             temp[2] = String.valueOf(dTotalError);
964             temp[3] = String.valueOf(kWh_CO2);
965             recursionResults.set(0, temp);
966         } else if (numRandAnnealing < anneallingValue) {
967             String temp[] = new String[4];
968             String t = "";
969             boolean first = true;
970             for (int d = 0; d < removedMetaRules.size(); d++) {
971                 if (first) {
972                     t = removedMetaRules.get(d).toString();
973                     first = false;
974                 } else {
975                     t = t.concat(", " + removedMetaRules.get(d));
976                 }
977             }
978             temp[0] = t;
979             temp[1] = String.valueOf(kWh);
980             temp[2] = String.valueOf(dTotalError);
981             temp[3] = String.valueOf(kWh_CO2);
982             recursionResults.set(0, temp);
983         }
984     }
985
986     recursiveIterator++;
987     if (removedMetaRules.size() == numrules) {
988         removedMetaRules.clear();
989     }
990     if (recursiveIterator == 15) {

```



```

992         solutionsFound = true;
993         removedMetaRules.clear();
994         String[] rs = recursionResults.get(0);
995         String[] splitted = rs[0].split(",");
996         for (int i = 0; i < splitted.length; i++) {
997             if (splitted[0].length() == 0)
998                 break;
999             removedMetaRules.add(Integer.valueOf(splitted[i]));
1000         }
1001     }
1002     calculateKWH5(sRecordsDaily);
1003     return public_ms;
1004     // return 0;
1005 } else {
1006     contiguous_ms = localContiguous_ms;
1007     dtTemp = dt1;
1008     GreenPlannerData.addAll(copiediotdatakwh);
1009     removedMetaRules.clear();
1010     recursiveIterator = 0;
1011     recursionResults.clear();
1012     solutionsFound = false;
1013
1014     public_ms = ms;
1015     total_daily_ms_CO2 += daily_ms_CO2;
1016     return public_ms;
1017 }
1018 }
1019
1020 }
1021 }
1022
1023 // END OF CALCULATE KWH METHOD - number of modifications
1024
1025 // GREEN PLANNER (CALCULATEkwh5 - associated recursion function)
1026 public void greenplanner(ArrayList<String[]> data, ArrayList<String[]> MRTrulesOriginal, double formaliseMaxError) {
1027     copiediotdata = new ArrayList<String[]>(data); // clone original dataset ArrayList
1028     MRTrules = new ArrayList<String[]>(MRTrulesOriginal);
1029     listRanges = MRTrules.toArray(new String[0][]);
1030     try {
1031         dtTemp = simpleFormatter.parse("2013-10-22 02:32:34.409"); // temp date
1032     } catch (Exception e) {
1033         System.out.println(e);
1034     }
1035     contiguous_ms = 0;
1036     ArrayList<String[]> recordsDaily = new ArrayList<String[]>();
1037     double dayChange = 22;
1038
1039     double total_ms = 0;
1040     GreenPlannerData = new ArrayList<String[]>();
1041     copiediotdata3 = new ArrayList<String[]>();
1042
1043     int monthchange = 0;
1044     int yearchange = 1998;
1045     DecimalFormat df1 = new DecimalFormat("#.##");
1046     df1.setRoundingMode(RoundingMode.CEILING);
1047     String monthString;
1048     boolean first = true;
1049
1050     try {
1051         for (int i = 0; i < copiediotdata.size(); i++) {
1052             String[] copiediotdataSplitHour = copiediotdata.get(i);
1053
1054             double tempDAYChange = getDay(copiediotdataSplitHour[0]);
1055             int tempmonthchange = getMonth(copiediotdataSplitHour[0]);
1056
1057             if (dayChange == tempDAYChange) {
1058                 if (i == copiediotdata.size() - 1) // only for the last record something...
1059                 {
1060                     recordsDaily.add(copiediotdata.get(i));
1061                     double hourly_ms = calculateKWH5(recordsDaily);
1062
1063                     total_ms = total_ms + hourly_ms;
1064                     recordsDaily.clear();
1065                 } else {
1066                     recordsDaily.add(copiediotdata.get(i));
1067                 }
1068             } else {
1069                 dayChange = tempDAYChange;
1070                 double hourly_ms = calculateKWH5(recordsDaily);
1071
1072                 total_ms = total_ms + hourly_ms;
1073                 recordsDaily.clear();
1074                 recordsDaily.add(copiediotdata.get(i));
1075             }
1076
1077             if (monthchange != tempmonthchange) {
1078                 monthchange = tempmonthchange;
1079
1080                 if (first) {
1081                     double seconds1 = total_ms / 1000;
1082                     double hours1 = seconds1 / 3600;

```

```

1082         double kWh1 = ((hours1 * 320) / 1000) * 1100;
1083
1084         kWhpermonth = kWhpermonth.concat(df1.format(kWh1));
1085         first = false;
1086     } else {
1087         double seconds1 = total_ms / 1000;
1088         double hours1 = seconds1 / 3600;
1089         double kWh1 = ((hours1 * 320) / 1000) * 1100;
1090
1091         kWhpermonth = kWhpermonth.concat(", " + df1.format(kWh1));
1092     }
1093 }
1094 }
1095 // END of calculation for the kwh per month
1096
1097 }
1098 } catch (Exception ex) {
1099
1100     System.out.println("First A: " + errorLine + " => Exception: " + ex + "\nAt line:\t");
1101     ex.printStackTrace();
1102 }
1103
1104 copiediotdata.clear();
1105 copiediotdata.addAll(GreenPlannerData);
1106 GreenPlannerData.clear();
1107
1108 double seconds = total_ms / 1000;
1109 double hours = seconds / 3600;
1110 double kWh = ((hours * 320) / 1000) * 1100;
1111 String label183 = "Total Kilowatt hour (kWh): " + Math.round(kWh) + " kWh";
1112 System.out.println(label183);
1113 totalkwh = Math.round(kWh);
1114 double seconds_CO2 = total_daily_ms_CO2 / 1000;
1115 double hours_CO2 = seconds_CO2 / 3600;
1116 double kWh_CO2 = ((hours_CO2 * 320) / 1000) * 1100; // 26
1117 DecimalFormat df = new DecimalFormat("#.##");
1118 df.setRoundingMode(RoundingMode.CEILING);
1119 String label176 = "Kilowatt hour (kWh) with CO2 emmissions: " + df.format(kWh - kWh_CO2) + " kWh";
1120 System.out.println(label176);
1121 kWh_with_CO2 = toDouble(df.format(kWh - kWh_CO2));
1122 String label177 = "Kilowatt hour (kWh) without CO2 emmissions: " + df.format(kWh_CO2) + " kWh";
1123 kWh_without_CO2 = toDouble(df.format(kWh_CO2));
1124 System.out.println(label177);
1125 double KgCO2_co2 = 0.449;
1126 String label179 = "CO2 emissions : " + df.format((kWh - kWh_CO2) * KgCO2_co2) + " kg CO2e";
1127
1128 CO2_kg_emissions = toDouble(df.format((kWh - kWh_CO2) * KgCO2_co2));
1129 System.out.println(label179);
1130 double dTotalError = 0;
1131 double dLineError = 0;
1132 double dTotalError_kWh = 0;
1133
1134 // *****SECOND AND LAST PHASE*****//
1135
1136 double maxError = 0;
1137 double minError = 100;
1138 boolean errorFound = false;
1139 int errorsize = 0;
1140
1141 try {
1142     for (int i = 0; i < copiediotdata.size(); i++) {
1143
1144         dLineError = 0; // gets the total record line error in regards of all the ranges
1145         for (int k = 0; k < listRanges.length; k++) {
1146             errorLine = String.join(" ", copiediotdata.get(i));
1147
1148             String[] copiediotdataSplitted2 = copiediotdata.get(i);
1149             String[] rangesSplitted = listRanges[k];
1150
1151             if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
1152             {
1153                 // checks time and temperature and if season is empty
1154                 if ((!rangesSplitted[0].equals("SUMMER") || !rangesSplitted[0].equals("WINTER"))
1155                     && rangesSplitted[3].equals("TEMPER.")) {
1156                     if (getHour(copiediotdataSplitted2[0]) >=
1157                         ruleHourFrom(rangesSplitted[0])
1158                         && getHour(copiediotdataSplitted2[0]) <=
1159                         ruleHourTo(rangesSplitted[0])
1160                         && toDouble(copiediotdataSplitted2[3]) >=
1161                         toDouble(rangesSplitted[1])
1162                         && toDouble(copiediotdataSplitted2[3]) <=
1163                         toDouble(rangesSplitted[2])) {
1164                         copiediotdata3.add(copiediotdata.get(i));
1165                         // its already on the ArrayList no need to add it twice
1166                     } else {
1167                         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1168                         if (getHour(copiediotdataSplitted2[0]) >=
1169                             ruleHourFrom(rangesSplitted[0])
1170                             && getHour(copiediotdataSplitted2[0]) <=

```



```

1172         ruleHourTo(rangesSplitted[0])
1173         && toDouble(copiediotdataSplitted2[3]) <
1174         toDouble(rangesSplitted[1])) {
1175         dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1176         - toDouble(copiediotdataSplitted2[3])) / 1);
1177         dLineError = dLineError + ((toDouble(rangesSplitted[1])
1178         - toDouble(copiediotdataSplitted2[3])) / 1);
1179
1180         if (dLineError > maxError) {
1181             maxError = dLineError;
1182         }
1183         if (dLineError < minError) {
1184             minError = dLineError;
1185         }
1186         errorFound = true;
1187     } else if (getHour(copiediotdataSplitted2[0]) >=
1188         ruleHourFrom(rangesSplitted[0])
1189         && getHour(copiediotdataSplitted2[0]) <=
1190         ruleHourTo(rangesSplitted[0])
1191         && toDouble(copiediotdataSplitted2[3]) >
1192         toDouble(rangesSplitted[2])) {
1193         dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1194         - toDouble(rangesSplitted[2])) / 1);
1195         dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1196         - toDouble(rangesSplitted[2])) / 1);
1197
1198         if (dLineError > maxError) {
1199             maxError = dLineError;
1200         }
1201         if (dLineError < minError) {
1202             minError = dLineError;
1203         }
1204         errorFound = true;
1205     }
1206 }
1207 }
1208 }
1209
1210 if ((rangesSplitted[0].equals("SUMMER") || rangesSplitted[0].equals("WINTER"))
1211     && rangesSplitted[3].equals("TEMPER.")) {
1212     int iSeason = getMonth(copiediotdataSplitted2[0]);
1213     // checks if SUMMER
1214     if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
1215         if (toDouble(copiediotdataSplitted2[3]) >=
1216
1217         toDouble(rangesSplitted[1])
1218         && toDouble(copiediotdataSplitted2[3]) <=
1219         toDouble(rangesSplitted[2])) {
1220         copiediotdata3.add(copiediotdata.get(i));
1221         // its already on the ArrayList no need to add it twice
1222     } else {
1223         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1224         if (toDouble(copiediotdataSplitted2[3]) <
1225         toDouble(rangesSplitted[1])) {
1226             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1227             - toDouble(copiediotdataSplitted2[3])) / 1);
1228             dLineError = dLineError + ((toDouble(rangesSplitted[1])
1229             - toDouble(copiediotdataSplitted2[3])) / 1);
1230
1231             if (dLineError > maxError) {
1232                 maxError = dLineError;
1233             }
1234             if (dLineError < minError) {
1235                 minError = dLineError;
1236             }
1237             errorFound = true;
1238         } else if (toDouble(copiediotdataSplitted2[3]) >
1239         toDouble(rangesSplitted[2])) {
1240             dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1241             - toDouble(rangesSplitted[2])) / 1);
1242             dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1243             - toDouble(rangesSplitted[2])) / 1);
1244
1245             if (dLineError > maxError) {
1246                 maxError = dLineError;
1247             }
1248             if (dLineError < minError) {
1249                 minError = dLineError;
1250             }
1251             errorFound = true;
1252         }
1253     }
1254 }
1255 // checks if WINTER
1256 else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
1257     if (toDouble(copiediotdataSplitted2[3]) >=
1258         toDouble(rangesSplitted[1])
1259         && toDouble(copiediotdataSplitted2[3]) <=
1260         toDouble(rangesSplitted[2])) {
1261         copiediotdata3.add(copiediotdata.get(i));

```

```

1262 // its already on the ArrayList no need to add it twice
1263 } else {
1264 // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1265 if (toDouble(copiediotdataSplitted2[3]) <
1266     toDouble(rangesSplitted[1])) {
1267     dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1268         - toDouble(copiediotdataSplitted2[3])) / 1);
1269     dLineError = dLineError + ((toDouble(rangesSplitted[1])
1270         - toDouble(copiediotdataSplitted2[3])) / 1);
1271
1272     if (dLineError > maxError) {
1273         maxError = dLineError;
1274     }
1275     if (dLineError < minError) {
1276         minError = dLineError;
1277     }
1278     errorFound = true;
1279 } else if (toDouble(copiediotdataSplitted2[3]) >
1280     toDouble(rangesSplitted[2])) {
1281     dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1282         - toDouble(rangesSplitted[2])) / 1);
1283     dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1284         - toDouble(rangesSplitted[2])) / 1);
1285
1286     if (dLineError > maxError) {
1287         maxError = dLineError;
1288     }
1289     if (dLineError < minError) {
1290         minError = dLineError;
1291     }
1292     errorFound = true;
1293 }
1294 }
1295 }
1296
1297 if (rangesSplitted[0].equals("kWh")) {
1298     if (toDouble(copiediotdataSplitted2[7]) <=
1299         toDouble(rangesSplitted[1])) {
1300         copiediotdata3.add(copiediotdata.get(i));
1301     } else {
1302         dTotalError_kWh = dTotalError_kWh + ((toDouble(copiediotdataSplitted2[7])
1303             - toDouble(rangesSplitted[1])) / 1);
1304         dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
1305             - toDouble(rangesSplitted[1])) / 1);
1306         dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
1307             - toDouble(rangesSplitted[1])) / 1);
1308         dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
1309             - toDouble(rangesSplitted[1])) / 1);
1310
1311         if (dLineError > maxError) {
1312             maxError = dLineError;
1313         }
1314         if (dLineError < minError) {
1315             minError = dLineError;
1316         }
1317         errorFound = true;
1318     }
1319 }
1320
1321 }
1322
1323 if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
1324 {
1325     // checks LIGHT
1326     if (rangesSplitted[3].equals("LIGHT")) {
1327         if (getHour(copiediotdataSplitted2[0]) >=
1328             ruleHourFrom(rangesSplitted[0])
1329             && getHour(copiediotdataSplitted2[0]) <=
1330                 ruleHourTo(rangesSplitted[0])
1331             && toInt(copiediotdataSplitted2[3]) >=
1332                 toInt(rangesSplitted[1])
1333             && toInt(copiediotdataSplitted2[3]) <=
1334                 toInt(rangesSplitted[2])) {
1335             copiediotdata3.add(copiediotdata.get(i));
1336             // its already on the ArrayList no need to add it twice
1337         } else {
1338             // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1339             if (getHour(copiediotdataSplitted2[0]) >=
1340                 ruleHourFrom(rangesSplitted[0])
1341                 && getHour(copiediotdataSplitted2[0]) <=
1342                     ruleHourTo(rangesSplitted[0])
1343                 && toDouble(copiediotdataSplitted2[3]) <
1344                     toDouble(rangesSplitted[1])) {
1345                 dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1346                     - toDouble(copiediotdataSplitted2[3])) / 1);
1347                 dLineError = dLineError + ((toDouble(rangesSplitted[1])
1348                     - toDouble(copiediotdataSplitted2[3])) / 1);
1349
1350                 if (dLineError > maxError) {
1351                     maxError = dLineError;

```

```

1352     }
1353     if (dLineError < minError) {
1354         minError = dLineError;
1355     }
1356     errorFound = true;
1357 } else if (getHour(copiediotdataSplitted2[0]) >=
1358     ruleHourFrom(rangesSplitted[0])
1359     && getHour(copiediotdataSplitted2[0]) <=
1360     ruleHourTo(rangesSplitted[0])
1361     && toDouble(copiediotdataSplitted2[3]) >
1362     toDouble(rangesSplitted[2])) {
1363     dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1364         - toDouble(rangesSplitted[2])) / 1);
1365     dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1366         - toDouble(rangesSplitted[2])) / 1);
1367
1368     if (dLineError > maxError) {
1369         maxError = dLineError;
1370     }
1371     if (dLineError < minError) {
1372         minError = dLineError;
1373     }
1374     errorFound = true;
1375 }
1376 }
1377 } // end of: checks LIGHT
1378
1379 // checks SUNNY
1380 if (rangesSplitted[3].equals("SUNNY")) {
1381     if (getHour(copiediotdataSplitted2[0]) >=
1382         ruleHourFrom(rangesSplitted[0])
1383         && getHour(copiediotdataSplitted2[0]) <=
1384         ruleHourTo(rangesSplitted[0])
1385         && toInt(copiediotdataSplitted2[3]) >=
1386         toInt(rangesSplitted[1])
1387         && toInt(copiediotdataSplitted2[3]) <=
1388         toInt(rangesSplitted[2])) {
1389         copiediotdata3.add(copiediotdata.get(i));
1390         // its already on the ArrayList no need to add it twice
1391     } else {
1392         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1393         if (toDouble(copiediotdataSplitted2[3]) <
1394             toDouble(rangesSplitted[1])) {
1395             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1396                 - toDouble(copiediotdataSplitted2[3])) / 1);
1397
1398             dLineError = dLineError + ((toDouble(rangesSplitted[1])
1399                 - toDouble(copiediotdataSplitted2[3])) / 1);
1400
1401             if (dLineError > maxError) {
1402                 maxError = dLineError;
1403             }
1404             if (dLineError < minError) {
1405                 minError = dLineError;
1406             }
1407             errorFound = true;
1408         } else if (toDouble(copiediotdataSplitted2[3]) >
1409             toDouble(rangesSplitted[2])) {
1410             dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1411                 - toDouble(rangesSplitted[2])) / 1);
1412             dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1413                 - toDouble(rangesSplitted[2])) / 1);
1414
1415             if (dLineError > maxError) {
1416                 maxError = dLineError;
1417             }
1418             if (dLineError < minError) {
1419                 minError = dLineError;
1420             }
1421             errorFound = true;
1422         }
1423     }
1424 } // end of :checks SUNNY
1425
1426 // checks CLOUDY
1427 if (rangesSplitted[3].equals("CLOUDY")) {
1428     if (getHour(copiediotdataSplitted2[0]) >=
1429         ruleHourFrom(rangesSplitted[0])
1430         && getHour(copiediotdataSplitted2[0]) <=
1431         ruleHourTo(rangesSplitted[0])
1432         && toInt(copiediotdataSplitted2[3]) >=
1433         toInt(rangesSplitted[1])
1434         && toInt(copiediotdataSplitted2[3]) <=
1435         toInt(rangesSplitted[2])) {
1436         copiediotdata3.add(copiediotdata.get(i));
1437         // its already on the ArrayList no need to add it twice
1438         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1439     } else if (toDouble(copiediotdataSplitted2[3]) <
1440         toDouble(rangesSplitted[1])) {
1441         dTotalError = dTotalError + ((toDouble(rangesSplitted[1])

```

```

1442         - toDouble(copiediotdataSplitted2[3])) / 1);
1443 dLineError = dLineError + ((toDouble(rangesSplitted[1])
1444     - toDouble(copiediotdataSplitted2[3])) / 1);
1445
1446     if (dLineError > maxError) {
1447         maxError = dLineError;
1448     } else if (dLineError < minError) {
1449         minError = dLineError;
1450     }
1451     errorFound = true;
1452 } else if (toDouble(copiediotdataSplitted2[3]) >
1453     toDouble(rangesSplitted[2])) {
1454
1455     dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1456         - toDouble(rangesSplitted[2])) / 1);
1457     dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1458         - toDouble(rangesSplitted[2])) / 1);
1459
1460     if (dLineError > maxError) {
1461         maxError = dLineError;
1462     } else if (dLineError < minError) {
1463         minError = dLineError;
1464     }
1465     errorFound = true;
1466 }
1467
1468 } // end of :checks CLOUDY
1469
1470 // checks DOOR OPEN-LIGHT
1471 if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
1472     if (getHour(copiediotdataSplitted2[0]) >=
1473         ruleHourFrom(rangesSplitted[0])
1474         && getHour(copiediotdataSplitted2[0]) <=
1475             ruleHourTo(rangesSplitted[0])
1476         && toInt(copiediotdataSplitted2[3]) >=
1477             toInt(rangesSplitted[1])
1478         && toInt(copiediotdataSplitted2[3]) <=
1479             toInt(rangesSplitted[2])) {
1480         copiediotdata3.add(copiediotdata.get(i));
1481         // its already on the ArrayList no need to add it twice
1482     } else {
1483         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1484         if (toDouble(copiediotdataSplitted2[3]) <
1485             toDouble(rangesSplitted[1])) {
1486             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1487
1488         - toDouble(copiediotdataSplitted2[3])) / 1);
1489         dLineError = dLineError + ((toDouble(rangesSplitted[1])
1490             - toDouble(copiediotdataSplitted2[3])) / 1);
1491
1492         if (dLineError > maxError) {
1493             maxError = dLineError;
1494         }
1495         if (dLineError < minError) {
1496             minError = dLineError;
1497         }
1498         errorFound = true;
1499     } else if (toDouble(copiediotdataSplitted2[3]) >
1500         toDouble(rangesSplitted[2])) {
1501         dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1502             - toDouble(rangesSplitted[2])) / 1);
1503         dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1504             - toDouble(rangesSplitted[2])) / 1);
1505
1506         if (dLineError > maxError) {
1507             maxError = dLineError;
1508         }
1509         if (dLineError < minError) {
1510             minError = dLineError;
1511         }
1512         errorFound = true;
1513     }
1514 }
1515 } // end of :checks DOOR OPEN-LIGHT
1516
1517 if (rangesSplitted[0].equals("kWh")) {
1518     if (toDouble(copiediotdataSplitted2[7]) <=
1519         toDouble(rangesSplitted[1])) {
1520         copiediotdata3.add(copiediotdata.get(i));
1521     } else {
1522         dTotalError_kWh = dTotalError_kWh + ((toDouble(copiediotdataSplitted2[7])
1523             - toDouble(rangesSplitted[1])) / 1);
1524         dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
1525             - toDouble(rangesSplitted[1])) / 1);
1526         dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
1527             - toDouble(rangesSplitted[1])) / 1);
1528
1529         if (dLineError > maxError) {
1530             maxError = dLineError;
1531         }
1532         if (dLineError < minError) {

```

```

1532         minError = dLineError;
1533     }
1534     errorFound = true;
1535 }
1536 } // end of kWh check
1537
1538 } // end of LS
1539
1540 }
1541
1542 if (errorFound == true) {
1543     errorsSize++;
1544     errorFound = false;
1545 }
1546
1547 }
1548 } catch (Exception e) {
1549     System.out.println("Second: " + errorLine);
1550 }
1551
1552 // *****end of second phase *****
1553
1554 df = new DecimalFormat("#.###");
1555 df.setRoundingMode(RoundingMode.CEILING);
1556
1557 dTotalError = dTotalError * 10;
1558
1559 String label189 = "Total Error: " + df.format((dTotalError * 100) / formaliseMaxError);
1560 String label186 = "Max Error: " + df.format((maxError * 100) / formaliseMaxError) + "%";
1561 String label185 = "Min Error: " + df.format((minError * 100) / formaliseMaxError) + "%";
1562 System.out.println(label189);
1563 System.out.println(label186);
1564 System.out.println(label185);
1565
1566 // calculate Total Average Error
1567 double averageErrorTotal = Math.round((dTotalError / copiediotdata.size()));
1568 String label182 = "Total Average Error: " + df.format((averageErrorTotal * 100) / formaliseMaxError) + "%";
1569 System.out.println(label182);
1570 totalaverageerror = toDouble(df.format((averageErrorTotal * 100) / formaliseMaxError));
1571 // calculate Budget Average Error
1572 double averageError_Budget = Math.round((dTotalError_kWh / copiediotdata.size()));
1573 String label180 = "Budget Average Error: " + df.format((averageError_Budget * 100) / formaliseMaxError) + "%";
1574 System.out.println(label180);
1575 budgeterror = toDouble(df.format((averageError_Budget * 100) / formaliseMaxError));
1576
1577 // calculate Convenience Average Error
1578 double averageError_Convenience = Math.round((dTotalError - dTotalError_kWh) / copiediotdata.size());
1579 String label181 = "Convenience Average Error: "
1580     + df.format((averageError_Convenience * 100) / formaliseMaxError) + "%";
1581 System.out.println(label181);
1582 convenienceerror = toDouble(df.format((averageError_Convenience * 100) / formaliseMaxError));
1583
1584 copiediotdata3.clear();
1585
1586 // *****SECOND RUN*****
1587 double meansSum = 0;
1588 maxError = 0;
1589 minError = 100;
1590 dTotalError = 0;
1591 dLineError = 0;
1592
1593 try {
1594     for (int i = 0; i < copiediotdata.size(); i++) {
1595         dLineError = 0; // gets the total record line error in regards of all the ranges
1596         for (int k = 0; k < listRanges.length; k++) {
1597             errorLine = String.join("|", copiediotdata.get(i));
1598
1599             String[] copiediotdataSplitted2 = copiediotdata.get(i);
1600             String[] rangesSplitted = listRanges[k];
1601
1602             if (copiediotdataSplitted2[2].startsWith("T")) // gets T for temperatures
1603             {
1604                 // checks time and temperature and if season is empty
1605                 if (!rangesSplitted[0].equals("SUMMER") || !rangesSplitted[0].equals("WINTER"))
1606                 {
1607                     && rangesSplitted[3].equals("TEMPER.") {
1608                         if (getHour(copiediotdataSplitted2[0]) >=
1609                             ruleHourFrom(rangesSplitted[0])
1610                             && getHour(copiediotdataSplitted2[0]) <=
1611                                 ruleHourTo(rangesSplitted[0])
1612                             && toDouble(copiediotdataSplitted2[3]) >=
1613                                 toDouble(rangesSplitted[1])
1614                             && toDouble(copiediotdataSplitted2[3]) <=
1615                                 toDouble(rangesSplitted[2])) {
1616                             copiediotdata3.add(copiediotdata.get(i));
1617                             // its already on the ArrayList no need to add it twice
1618                         } else {
1619                             // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1620

```



```

1622 if (getHour(copiediotdataSplitted2[0]) >=
1623     ruleHourFrom(rangesSplitted[0])
1624     && getHour(copiediotdataSplitted2[0]) <=
1625         ruleHourTo(rangesSplitted[0])
1626     && toDouble(copiediotdataSplitted2[3]) <
1627         toDouble(rangesSplitted[1])) {
1628     dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1629         - toDouble(copiediotdataSplitted2[3])) / 1);
1630     dLineError = dLineError + ((toDouble(rangesSplitted[1])
1631         - toDouble(copiediotdataSplitted2[3])) / 1);
1632
1633     if (dLineError > maxError) {
1634         maxError = dLineError;
1635     }
1636     if (dLineError < minError) {
1637         minError = dLineError;
1638     }
1639 } else if (getHour(copiediotdataSplitted2[0]) >=
1640     ruleHourFrom(rangesSplitted[0])
1641     && getHour(copiediotdataSplitted2[0]) <=
1642         ruleHourTo(rangesSplitted[0])
1643     && toDouble(copiediotdataSplitted2[3]) >
1644         toDouble(rangesSplitted[2])) {
1645     dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1646         - toDouble(rangesSplitted[2])) / 1);
1647     dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1648         - toDouble(rangesSplitted[2])) / 1);
1649
1650     if (dLineError > maxError) {
1651         maxError = dLineError;
1652     }
1653     if (dLineError < minError) {
1654         minError = dLineError;
1655     }
1656 }
1657 }
1658 }
1659 }
1660 }
1661 }
1662 }
1663 if ((rangesSplitted[0].equals("SUMMER") || rangesSplitted[0].equals("WINTER"))
1664     && rangesSplitted[3].equals("TEMPER.")) {
1665     int iSeason = getMonth(copiediotdataSplitted2[0]);
1666

```

```

1667 // checks if SUMMER
1668 if ((iSeason == 6) || (iSeason == 7) || (iSeason == 8)) {
1669     if (toDouble(copiediotdataSplitted2[3]) >=
1670         toDouble(rangesSplitted[1])
1671         && toDouble(copiediotdataSplitted2[3]) <=
1672             toDouble(rangesSplitted[2])) {
1673         copiediotdata3.add(copiediotdata.get(i));
1674         // its already on the ArrayList no need to add it twice
1675     } else {
1676         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1677         if (toDouble(copiediotdataSplitted2[3]) <
1678             toDouble(rangesSplitted[1])) {
1679             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1680                 - toDouble(copiediotdataSplitted2[3])) / 1);
1681             dLineError = dLineError + ((toDouble(rangesSplitted[1])
1682                 - toDouble(copiediotdataSplitted2[3])) / 1);
1683
1684             if (dLineError > maxError) {
1685                 maxError = dLineError;
1686             }
1687             if (dLineError < minError) {
1688                 minError = dLineError;
1689             }
1690         } else if (toDouble(copiediotdataSplitted2[3]) >
1691             toDouble(rangesSplitted[2])) {
1692             dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1693                 - toDouble(rangesSplitted[2])) / 1);
1694             dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1695                 - toDouble(rangesSplitted[2])) / 1);
1696
1697             if (dLineError > maxError) {
1698                 maxError = dLineError;
1699             }
1700             if (dLineError < minError) {
1701                 minError = dLineError;
1702             }
1703         }
1704     }
1705 }
1706 // checks if WINTER
1707 else if ((iSeason == 12) || (iSeason == 1) || (iSeason == 2)) {
1708     if (toDouble(copiediotdataSplitted2[3]) >=
1709

```

```

1712         toDouble(rangesSplitted[1])
1713         && toDouble(copiediotdataSplitted2[3]) <=
1714             toDouble(rangesSplitted[2])) {
1715             copiediotdata3.add(copiediotdata.get(i));
1716             // its already on the ArrayList no need to add it twice
1717
1718     } else {
1719         // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1720         if (toDouble(copiediotdataSplitted2[3]) <
1721             toDouble(rangesSplitted[1])) {
1722             dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1723                 - toDouble(copiediotdataSplitted2[3])) / 1);
1724             dLineError = dLineError + ((toDouble(rangesSplitted[1])
1725                 - toDouble(copiediotdataSplitted2[3])) / 1);
1726
1727             if (dLineError > maxError) {
1728                 maxError = dLineError;
1729             }
1730             if (dLineError < minError) {
1731                 minError = dLineError;
1732             }
1733
1734         } else if (toDouble(copiediotdataSplitted2[3]) >
1735             toDouble(rangesSplitted[2])) {
1736             dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1737                 - toDouble(rangesSplitted[2])) / 1);
1738             dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1739                 - toDouble(rangesSplitted[2])) / 1);
1740
1741             if (dLineError > maxError) {
1742                 maxError = dLineError;
1743             }
1744             if (dLineError < minError) {
1745                 minError = dLineError;
1746             }
1747         }
1748     }
1749 }
1750
1751 }
1752
1753 }
1754
1755 if (rangesSplitted[0].equals("kWh")) {
1756     if (toDouble(copiediotdataSplitted2[7]) <=
        toDouble(rangesSplitted[4])) {
            copiediotdata3.add(copiediotdata.get(i));
        } else {
            dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
                - toDouble(rangesSplitted[2])) / 1);
            dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
                - toDouble(rangesSplitted[2])) / 1);
            if (dLineError > maxError) {
                maxError = dLineError;
            }
            if (dLineError < minError) {
                minError = dLineError;
            }
            errorFound = true;
        }
    }
}
}
}
if (copiediotdataSplitted2[2].startsWith("LS")) // gets LS for light sensor
{
    // checks LIGHT
    if (rangesSplitted[3].equals("LIGHT")) {
        if (getHour(copiediotdataSplitted2[0]) >=
            ruleHourFrom(rangesSplitted[0])
            && getHour(copiediotdataSplitted2[0]) <=
                ruleHourTo(rangesSplitted[0])
            && toInt(copiediotdataSplitted2[3]) >=
                toInt(rangesSplitted[1])
            && toInt(copiediotdataSplitted2[3]) <=
                toInt(rangesSplitted[2])) {
            copiediotdata3.add(copiediotdata.get(i));
            // its already on the ArrayList no need to add it twice
        } else {
            // assigns TOTAL ERROR AND CURRENT RECORD ERROR
            if (getHour(copiediotdataSplitted2[0]) >=
                ruleHourFrom(rangesSplitted[0])
                && getHour(copiediotdataSplitted2[0]) <=
                    ruleHourTo(rangesSplitted[0])
                && toDouble(copiediotdataSplitted2[3]) <
                    toDouble(rangesSplitted[1])) {
                dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
                    - toDouble(copiediotdataSplitted2[3])) / 1);
                dLineError = dLineError + ((toDouble(rangesSplitted[1])

```

```

1802         - toDouble(copiediotdataSplitted2[3])) / 1);
1803
1804         if (dLineError > maxError) {
1805             maxError = dLineError;
1806         }
1807         if (dLineError < minError) {
1808             minError = dLineError;
1809         }
1810     } else if (getHour(copiediotdataSplitted2[0]) >=
1811         ruleHourFrom(rangesSplitted[0])
1812         && getHour(copiediotdataSplitted2[0]) <=
1813             ruleHourTo(rangesSplitted[0])
1814         && toDouble(copiediotdataSplitted2[3]) >
1815             toDouble(rangesSplitted[2])) {
1816         dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1817             - toDouble(rangesSplitted[2])) / 1);
1818         dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1819             - toDouble(rangesSplitted[2])) / 1);
1820
1821         if (dLineError > maxError) {
1822             maxError = dLineError;
1823         }
1824         if (dLineError < minError) {
1825             minError = dLineError;
1826         }
1827     }
1828 }
1829 } // end of: checks LIGHT
1830
1831 // checks SUNNY
1832 if (rangesSplitted[3].equals("SUNNY")) {
1833     if (getHour(copiediotdataSplitted2[0]) >=
1834         ruleHourFrom(rangesSplitted[0])
1835         && getHour(copiediotdataSplitted2[0]) <=
1836             ruleHourTo(rangesSplitted[0])
1837         && toInt(copiediotdataSplitted2[3]) >=
1838             toInt(rangesSplitted[1])
1839         && toInt(copiediotdataSplitted2[3]) <=
1840             toInt(rangesSplitted[2])) {
1841         copiediotdata3.add(copiediotdata.get(i));
1842         // its already on the ArrayList no need to add it twice
1843     } else {

```

```

1847 // assigns TOTAL ERROR AND CURRENT RECORD ERROR
1848 if (toDouble(copiediotdataSplitted2[3]) <
1849     toDouble(rangesSplitted[1])) {
1850     dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
1851         - toDouble(copiediotdataSplitted2[3])) / 1);
1852     dLineError = dLineError + ((toDouble(rangesSplitted[1])
1853         - toDouble(copiediotdataSplitted2[3])) / 1);
1854
1855     if (dLineError > maxError) {
1856         maxError = dLineError;
1857     }
1858     if (dLineError < minError) {
1859         minError = dLineError;
1860     }
1861 } else if (toDouble(copiediotdataSplitted2[3]) >
1862     toDouble(rangesSplitted[2])) {
1863     dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
1864         - toDouble(rangesSplitted[2])) / 1);
1865     dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
1866         - toDouble(rangesSplitted[2])) / 1);
1867
1868     if (dLineError > maxError) {
1869         maxError = dLineError;
1870     }
1871     if (dLineError < minError) {
1872         minError = dLineError;
1873     }
1874 }
1875 }
1876 } // end of :checks SUNNY
1877
1878 // checks CLOUDY
1879 if (rangesSplitted[3].equals("CLOUDY")) {
1880     if (getHour(copiediotdataSplitted2[0]) >=
1881         ruleHourFrom(rangesSplitted[0])
1882         && getHour(copiediotdataSplitted2[0]) <=
1883             ruleHourTo(rangesSplitted[0])
1884         && toInt(copiediotdataSplitted2[3]) >=
1885             toInt(rangesSplitted[1])
1886         && toInt(copiediotdataSplitted2[3]) <=
1887             toInt(rangesSplitted[2])) {
1888         copiediotdata3.add(copiediotdata.get(i));
1889         // its already on the ArrayList no need to add it twice
1890     } else {

```



```

1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981

```

```

    } else if (toDouble(copiediotdataSplitted2[3]) <
        toDouble(rangesSplitted[1])) {
        // assigns TOTAL ERROR AND CURRENT RECORD ERROR
        dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
            - toDouble(copiediotdataSplitted2[3])) / 1);
        dLineError = dLineError + ((toDouble(rangesSplitted[1])
            - toDouble(copiediotdataSplitted2[3])) / 1);

        if (dLineError > maxError) {
            maxError = dLineError;
        } else if (dLineError < minError) {
            minError = dLineError;
        }
    }

    } else if (toDouble(copiediotdataSplitted2[3]) >
        toDouble(rangesSplitted[2])) {
        dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
            - toDouble(rangesSplitted[2])) / 1);
        dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
            - toDouble(rangesSplitted[2])) / 1);

        if (dLineError > maxError) {
            maxError = dLineError;
        } else if (dLineError < minError) {
            minError = dLineError;
        }
    }
}

} // end of :checks CLOUDY

// checks DOOR OPEN-LIGHT
if (rangesSplitted[3].equals("DOOR OPEN-LIGHT")) {
    if (getHour(copiediotdataSplitted2[0]) >=
        ruleHourFrom(rangesSplitted[0])
        && getHour(copiediotdataSplitted2[0]) <=
            ruleHourTo(rangesSplitted[0])
        && toInt(copiediotdataSplitted2[3]) >=
            toInt(rangesSplitted[1])
        && toInt(copiediotdataSplitted2[3]) <=
            toInt(rangesSplitted[2])) {
        copiediotdata3.add(copiediotdata.get(i));
        // its already on the ArrayList no need to add it twice
    }
}

} else {
    // assigns TOTAL ERROR AND CURRENT RECORD ERROR
    if (toDouble(copiediotdataSplitted2[3]) <
        toDouble(rangesSplitted[1])) {
        dTotalError = dTotalError + ((toDouble(rangesSplitted[1])
            - toDouble(copiediotdataSplitted2[3])) / 1);
        dLineError = dLineError + ((toDouble(rangesSplitted[1])
            - toDouble(copiediotdataSplitted2[3])) / 1);

        if (dLineError > maxError) {
            maxError = dLineError;
        }
        if (dLineError < minError) {
            minError = dLineError;
        }
    }

    } else if (toDouble(copiediotdataSplitted2[3]) >
        toDouble(rangesSplitted[2])) {
        dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[3])
            - toDouble(rangesSplitted[2])) / 1);
        dLineError = dLineError + ((toDouble(copiediotdataSplitted2[3])
            - toDouble(rangesSplitted[2])) / 1);

        if (dLineError > maxError) {
            maxError = dLineError;
        }
        if (dLineError < minError) {
            minError = dLineError;
        }
    }
}

} // end of :checks DOOR OPEN-LIGHT

if (rangesSplitted[0].equals("kWh")) {
    if (toDouble(copiediotdataSplitted2[7]) <=
        toDouble(rangesSplitted[1])) {
        copiediotdata3.add(copiediotdata.get(i));
    } else {
        dTotalError = dTotalError + ((toDouble(copiediotdataSplitted2[7])
            - toDouble(rangesSplitted[2])) / 1);
        dLineError = dLineError + ((toDouble(copiediotdataSplitted2[7])
            - toDouble(rangesSplitted[2])) / 1);

        if (dLineError > maxError) {

```

```

1982         maxError = dLineError;
1983     }
1984     if (dLineError < minError) {
1985         minError = dLineError;
1986     }
1987     errorFound = true;
1988 }
1989 } // end of kWh check
1990
1991 } // end of LS
1992
1993 }
1994
1995     meansSum = meansSum + Math.pow(dLineError - averageErrorTotal, 2);
1996
1997 }
1998 } catch (Exception e) {
1999     System.out.println("Second: " + errorLine);
2000 }
2001
2002
2003 double standardDeviation = Math.sqrt(meansSum / copiediotdata.size());
2004 String labell84 = "Standard Deviation: " + df.format((standardDeviation * 100) / formaliseMaxError) + "%";
2005 deviation = toDouble(df.format((standardDeviation * 100) / formaliseMaxError));
2006
2007 copiediotdata3.clear();
2008
2009 postdata();
2010 }// END OF Green PLANNER - number of modifications
2011
2012 }

```