TUGAS AKHIR – EE 184801

DESAIN DAN IMPLEMENTASI *SMART HOME* KONSUMSI DAYA RENDAH MENGGUNAKAN ALGORITMA OPTIMISASI *CUCKOO-EARTHWORM*

Miftahul Falahi Alfafa
NRP 07111745000082

Dosen Pembimbing
Yusuf Bilfaqih, ST., MT.

DEPARTEMEN TEKNIK ELEKTRO
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember
Surabaya 2020

-----Halaman ini sengaja dikosongkan-----

FINAL PROJECT **– EE 184801**

DESIGN AND IMPLEMENTATION OF LOW POWER CONSUMPTION SMART HOME USING CUCKOO-EARTHWORM OPTIMIZATION ALGORITHM

Miftahul Falahi Alfafa
NRP 07111745000082

Supervisor
Yusuf Bilfaqih, ST., MT.

DEPARTMENT OF ELECTRICAL ENGINEERING
Faculty of Intelligent Electrical and Informatics Technology
Institut Teknologi Sepuluh Nopember
Surabaya 2020

-----Halaman ini sengaja dikosongkan-----

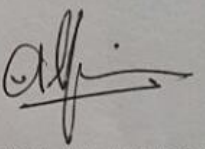
PERNYATAAN KEASLIAN TUGAS AKHIR

Dengan ini saya menyatakan bahwa isi sebagian maupun keseluruhan Tugas Akhir saya dengan judul "**Desain dan Implementasi *Smart Home* Daya Rendah menggunakan Algoritma Optimisasi *Cuckoo-Earthworm***" adalah benar-benar hasil karya intelektual mandiri, diselesaikan tanpa menggunakan bahan-bahan yang tidak diijinkan dan bukan merupakan karya pihak lain yang saya akui sebagai karya sendiri.

Semua referensi yang dikutip maupun dirujuk telah ditulis secara lengkap pada daftar pustaka.

Apabila ternyata pernyataan ini tidak benar, saya bersedia menerima sanksi sesuai peraturan yang berlaku.

Surabaya, 13 Januari 2020



Miftahul Falahi Alfafa
NRP 07111745000082

-----Halaman ini sengaja dikosongkan----

DESAIN DAN IMPLEMENTASI *SMART HOME* KONSUMSI DAYA RENDAH MENGGUNAKAN ALGORITMA OPTIMISASI *CUCKOO-EARTHWORM*

TUGAS AKHIR

Diajukan Guna Memenuhi Sebagian Persyaratan
Untuk Memperoleh Gelar Sarjana Teknik
Pada
Bidang Studi Teknik Sistem Pengaturan
Departemen Teknik Elektro
Institut Teknologi Sepuluh Nopember

Menyetujui:

Dosen Pembimbing,

Yusuf Bilfaqih, S.T., M.T.
NIP. 197203251999031001

KEMENTERIAN PENDIDIKAN DAN KEBUDAYAAN
INSTITUT TEKNOLOGI SEPULUH NOPEMBER
FAKULTAS TEKNOLOGI ELEKTRO DAN INFORMATIKA CERDAS
DEPARTEMEN TEKNIK ELEKTRO

SURABAYA
DESEMBER, 2020

-----Halaman ini sengaja dikosongkan-----

DESAIN DAN IMPLEMENTASI *SMART HOME* KONSUMSI DAYA RENDAH MENGGUNAKAN ALGORITMA OPTIMISASI *CUCKOO-EARTHWORM*

Nama : Miftahul Falahi Alfafa
Pembimbing : Yusuf Bilfaqih, ST., MT.

ABSTRAK

Penggunaan energi merupakan hal yang paling penting pada sistem *smart home*, karena dengan energi yang kecil maka sistem *smart home* akan semakin efisien dan juga ekonomis. Konsumsi energi pada sistem *smart home* dioptimalkan dengan teknik penjadwalan peralatan secara *real time* berdasarkan harga listrik pada saat itu dan persentase kenyamanan pengguna, semakin besar persentase kenyamanan pengguna maka semakin maksimal pula suatu peralatan rumah tangga dalam memberikan kenyamanan pada pengguna yang berarti penggunaan daya juga semakin besar pula. Algoritma *Cuckoo-Earthworm* digunakan untuk proses penjadwalan peralatan secara *real time*. Sistem *smart home* dilengkapi dengan Raspberry Pi3 sebagai HUB *controller* dan *Smart Plug* yang digunakan untuk monitor energi yang digunakan pada setiap peralatan dan juga sebagai *switch* untuk melakukan penjadwalan. Komunikasi antara HUB *controller* dan setiap *device* menggunakan jaringan Z-Wave. Untuk *user interface* menggunakan *Home Assistant*. Pada implemetasi sistem *smart home* dengan daya rendah menggunakan algoritma *Cuckoo-Eartworm* kali ini didapatkan pengurangan biaya mencapai 11.24% dan energi mencapai 32.52% dari peralatan yang tidak terjadwal pada tingkat kenyamanan terendah.

Kata Kunci: *Smart Home*, *Cuckoo-Earthworm*, *Smart Plug*, Raspberry Pi3, Z-Wave, *Home Assistant*

-----Halaman ini sengaja dikosongkan-----

DESIGN AND IMPLEMENTATION OF LOW POWER CONSUMPTION SMART HOME USING CUCKOO-EARTHWORM OPTIMIZATION ALGORITHM

Student's name : Miftahul Falahi Alfafa
Supervisor : Yusuf Bilfaqih, ST., MT.

ABSTRACT

Utilization of energy is an important thing in a Smart Home system, because with the small energy then the Smart Home system will be more efficient and economic. Energy concumption on Smart Home is optimized with real time scheduling technique on home appliances based on current electricity price and user comfort percentage, the greater the user's comfort percentage, the more maximum of an appliance to give convenience to a user that means the greater the power consumption of each appliance. Cuckoo-Earthworm algorithm is used for real time appliances scheduling. Smart Home system is equipped by Raspberry Pi3 as a HUB controller and Smart Plug that is used for energy monitoring for each appliance and as a switch to do scheduling. Communication between HUB controller and each appliance uses Z-Wave network. For user interface uses Home Assistant. In this implementation low power Smart Home system using Cuckoo-Earthworm is obtained the reduction of cost by 11.24% and energy up to 32.52% instead of non-scheduling appliances at the lowest comfort level.

Keywords: *Smart Home, Cuckoo-Earthworm, Smart Plug, Raspberry Pi3, Z-Wave, Home Assistant*

-----Halaman ini sengaja dikosongkan-----

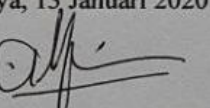
KATA PENGANTAR

Puji syukur penulis panjatkan kehadirat Allah SWT yang selalu memberikan rahmat dan hidayah-Nya sehingga tugas akhir ini dapat terselesaikan dengan baik. Shalawat serta salam semoga selalu dilimpahkan kepada Rasulullah Muhammad SAW, keluarga, sahabat, dan umat muslim yang senantiasa meneladani beliau. Tugas akhir ini disusun bertujuan untuk memenuhi sebagian persyaratan guna menyelesaikan pendidikan S1 pada Bidang Studi Teknik Sistem Pengaturan, Departemen Teknik Elektro, Fakultas Teknologi Elektro dan Informatika Cerdas, Institut Teknologi Sepuluh Nopember Surabaya yang berjudul **"Desain dan Implementasi *Smart Home* Daya Rendah menggunakan Algoritma Optimisasi *Cuckoo-Earthworm*"** atas selesainya penyusunan tugas akhir ini, penulis mengucapkan terima kasih kepada:

1. Ibu dan Bapak penulis yang memberikan berbagai bentuk doa serta dukungan tulus tiada henti, dalam keadaan apapun. Semoga Allah selalu memberikan kesehatan.
2. Bapak Yusuf Bilfaqih, ST., MT. selaku Dosen Pembimbing atas segala bimbingan ilmu, moral, dan spiritual dari awal hingga terselesaikannya tugas akhir,
3. Seluruh Staf/Karyawan/Dosen Departemen Teknik Elektro yang telah memberikan banyak ilmu dan menciptakan suasana belajar yang mendukung.
4. Teman-teman yang selalu mendukung saya secara ilmu, moral dan spiritual untuk menyelesaikan Tugas Akhir.

Penulis berharap semoga tugas akhir ini dapat bermanfaat dalam pengembangan keilmuan di kemudian hari.

Surabaya, 13 Januari 2020



Penulis

-----Halaman ini sengaja dikosongkan-----

DAFTAR ISI

DAFTAR GAMBAR

DAFTAR TABEL

-----Halaman ini sengaja dikosongkan-----

BAB I
PENDAHULUAN

1.1 Latar Belakang

Penggunaan sistem *smart home* sudah menjadi tren untuk era digital seperti sekarang ini. Dengan adanya sistem *smart home* maka pengontrolan dan *monitoring* penggunaan peralatan rumah tangga menjadi sangat mudah karena bisa dilakukan dimana saja dan kapan saja. Namun sistem *smart home* biasa hanya bisa mempermudah pengguna dalam hal pengontrolan dan monitoring peralatan rumah tangga saja tanpa bisa melakukan optimasi pada daya listrik yang dikonsumsi secara otomatis akibatnya pengaturan kebutuhan konsumsi daya lsitrik masih bergantung pada pengguna.

Hal tersebut sangat diperlukan perhatian lebih oleh penghuni rumah, karena walaupun seluruh peralatan rumah tangga yang terhubung ke sistem *smart home* bisa dikontrol dan di-*monitoring* dari mana saja dan kapan saja tetapi hal tersebut tidak bisa untuk di-*monitoring* secara terus-menerus dan kemungkinan lepas dari pengamatan pasti ada dimana saat penghuni rumah lupa untuk mematikan suatu peralatan tertentu seperti lampu taman, *spin drayer*, *washing machine*, lampu kamar tidur, dan lain-lain. Dari hal tersebut maka proses penghematan konsumsi daya pada sistem *smart home* masih belum optimal dan perlu perhatian khusus dari penghuni rumah.

1.2 Permasalahan

Terdapat sistem *smart home* yang belum mengunakan optimisasi penggunaan daya. Sehingga pemakaian energi listrik menjadi tak terkendali yang mengakibatkan tingginya biaya yang harus dibayarkan.

1.3 Tujuan

Penelitian tugas akhir ini bertujuan untuk:

1. Mendesain dan mengimplementasikan sistem *smart home* untuk mengoptimalkan penggunaan daya.
2. Menguji keluaran daya dan biaya pada sistem *smart home* yang telah dioptimisasi.

1.4 Batasan Masalah

Batasan masalah dalam tugas akhir ini adalah:

1. Membuat dan mengimplementasikan sistem *smart home* untuk rumah dengan penggunaan daya yang besar (*home industry*)
2. Hub *controller* dibuat dengan Raspberry Pi3, Aeotec USB Stick dan *Home Assistant*
3. *Power monitoring* dengan menggunakan Smart Plug Neo Coolcam
4. Algoritma yang digunakan untuk penjadwalan adalah *Cuckoo-Earthworm*

1.5 Metodologi

Metodologi yang digunakan dalam menyusun penelitian tugas akhir ini adalah sebagai berikut:

1. Studi Literatur
   Mengumpulkan referensi-referensi yang berhubungan dengan *smart home* daya rendah, algoritma *Cuckoo-Earthworm* seperti buku, *paper*, serta jurnal
2. Perancangan
   Tahap ini berisi proses perancangan peralatan *smart home* meliputi perancangan konsep dan detail menggunakan algoritma *Cuckoo-Earthworm* berdasarkan harga listrik saat itu dan tingkat kenyamanan pengguna
3. Analisis Data dan Hasil Simulasi
   Memberikan kesimpulan mengenai studi sistem *smart home* daya rendah pada proses penghematan daya berupa teknik penjadwalan
4. Penyelesaian Laporan Tugas Akhir
   Tahap ini dilakukan sebagai tahap akhir dari serangkaian pengerjaan tugas akhir ini. Juga dilakukan guna memenuhi persyaratan kelulusan mata kuliah Tugas Akhir

1.6 Sistematika Penulisan

Sistematika penulisan dalam tugas akhir ini terdiri dari lima BAB dengan uraian sebagai berikut:

1. BAB 1 pendahuluan menjelaskan latar belakang masalah, tujuan, metodologi, sistematika penulisan, dan relevansi
2. BAB 2 membahas kajian pustaka dan dasar teori yang membahas mengenai teori-teori penunjang yang berkaitan dengan sistem smart home daya rendah, dan algoritma *Cuckoo-Earthworm*
3. BAB 3 menyajikan rancangan sistem *smart home* daya rendah
4. BAB 4 membahas pengujian dan analisis data, yaitu membahas tentang simulasi sistem *smart home* daya rendah dengan teknik

penjadwalan peralatan berdasarkan harga listrik dan tingkat kenyamanan pengguna

5. BAB 5 menyampaikan kesimpulan dan saran

1.7 Relevansi

Penelitian diharapkan dapat memberikan manfaat, yaitu:

1. Menjadi referensi yang dapat menunjang pembuatan sistem *smart home* yang dapat mengoptimalkan penggunaan daya dan andal.
2. Menjadi referensi bagi mahasiswa yang akan mengerjakan penelitian dengan topik sistem *smart home* daya rendah.

BAB II
TINJAUAN PUSTAKA

Beberapa teori penunjang yang dipaparkan dalam buku tugas akhir ini adalah teori dasar mengenai antara lain *Smart home*, Raspberry Pi3, USB Z-Wave Stick, Z-Wave, Smart Plug Neo Coolcam, Algoritma *Cuckoo-Earthworm*, *Home Assistant*.

2.1 *Smart Home*

Smart home adalah aplikasi gabungan antara teknologi dan pelayanan yang dikhususkan pada lingkungan rumah dengan fungsi tertentu yang bertujuan meningkatkan keamanan, efisiensi dan kenyamanan penghuninya. Sistem *smart home* biasanya terdiri dari perangkat *monitoring*, perangkat kontrol dan otomatis ada beberapa perangkat yang dapat diakses menggunakan komputer. *Smart home* merupakan sebuah sistem yang dirancang dengan bantuan komputer yang akan memberikan kenyamanan, keamanan dan penghematan energi yang berlangsung secara otomatis sesuai dengan kendali pengguna dan terprogram melalui komputer pada gedung atau tempat tinggal kita. Teknologi yang dirancang untuk *smart home* ini bertujuan untuk memudahkan pemilik rumah dalam memantau kondisi peralatan elektronik yang terhubung dari gadget yang dimiliki.

Gambar 2. 1 *Smart Home*

2.2 Raspberry Pi3

Raspberry Pi3 merupakan komputer mini yang dapat dihubungkan ke *monitor* atau TV dan menggunakan *mouse* dan *keyboard* standar. Raspberry dapat digunakan seperti komputer desktop pada umumnya seperti untuk *browsing internet*, *programming*, bermain *game*, dan lain-lain. Versi Raspberry yang terakhir adalah Raspberry Pi3 b+ yang memiliki CPU Quad-core 64-bit ARM Cortex A53 1.2 GHz, 1GB SDRAM.

Gambar 2. 2 Raspberry Pi3 b+

2.3 Z-Wave

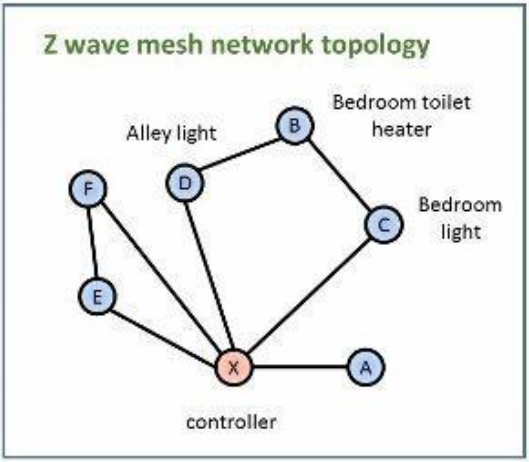
Z-Wave merupakan protokol komunikasi *wireless* yang menggunakan gelombang radio dengan penggunaan energi yang rendah. Z-Wave umumnya digunakan untuk *home automation*. Jaringan Z-Wave menggunakan topologi *mesh*. Dengan adanya Z-Wave, semua peralatan yang ada dirumah dapat dikontrol dengan gelombang radio dari jarak jauh.

Z-Wave didesain untuk menyediakan transmisi yang andal dengan *delay* latensi yang rendah untuk pengiriman paket dengan sedikit data dengan kecepatan pengiriman mencapai 100kbit/s. Dengan demikian Z-Wave sangat cocok untuk keperluan kontrol dan aplikasi yang behubungan dengan *sensor*. Jarak komunikasi antara dua *node* yaitu sekitar 30 meter dan kemampuan data pesan yang dapat melompat sampai 4 kali diantara *node*. Hal ini membuat cakupan yang cukup untuk kebanyakan gedung atau rumah. Dalam setiap negara memiliki aturan yang berbeda terhadap penggunaan frekuensi dari Z-Wave. Hal ini disebabkan karena frekuensi dari Z-Wave sama dengan beberapa peralatan elektronik dan telpon tanpa kabel. Berikut ini nilai frekuensi dan negara yang boleh menggunakan:

Tabel 2. 1 Frekuensi Z-Wave di setiap negara

Frekuensi (MHz)	**Negara**
865.2	India
869	Russia
868.4	China, Singapore, South Africa

868.40, 868.42, 869.85	CEPT Countries (Europe and other countries in region), French Guiana, Indonesia
908.4, 908.42, 916	USA, Canada, Argentina, Guatemala, The Bahamas, Jamaica, Barbados, Mexico, Bermuda, Nicaragua, Bolivia, Panama, British Virgin Islands, Suriname, Cayman Islands, Trinidad & Tobago, Colombia, Turks & Caicos, Ecuador, Uruguay
916	Israel
919.8	Hong Kong
919.8, 921.4	Australia, New Zealand, Malaysia, Brazil, Chile, El Salvador, Peru
919 - 923	South Korea
920 - 923	Thailand
920 - 925	Taiwan
922 - 926	Japan



Gambar 2. 3 Z-Wave *topology*

2.4 USB Z-Wave Stick

USB Z-Wave Stick merupakan komponen yang digunakan untuk membuat *gateway*. *Gateway* berfungsi sebagai kontroler untuk tiap-tiap *node* pada jaringan dengan topologi *mesh*. Dengan adanya *gateway* maka masing-masing *node* dapat masuk ke jaringan *mesh* tersebut melalui sebuah instruksi *join* pada setiap *device*. Denga adanya USB Z-Wave Stick maka status dari masing-masing *node* dapat dimonitor dan dikontrol. Data dari masing-masing *node* akan dikirim ke USB Z-Wave Stick yang kemudian diteruskan ke mini PC untuk ditampilkan dalam bentuk grafik maupun untuk dianalisis.

Gambar 2. 4 Z-Wave USB Stick

2.5 *Smart Plug*

Smart Plug merupakan device yang menghubungkan berbagai peralatan ke sumber listrik AC. Dengan adanya *smart plug*, setiap peralatan yang terhubung ke *smart plug* dapat dikontrol dan dimonitor dari jarak jauh melalui sebuah aplikasi *smart home*. Hal tersebut sangat memudahkan jika ingin melihat jumlah daya yang digunakan pada masing-masing peralatan maupun untuk menyalakan atau mematikan setiap peralan rumah tangga.

Gambar 2. 5 *Smart Plug*

2.6 *Home Assistant*

Home Assistant merupakan aplikasi untuk mengontrol dan memonitor peralatan rumah tangga secara *realtime* dengan menggunakan media gelombang radio dan dibuat dengan bahasa pemrograman Python yang dapat dijalankan kedalam berbagai macam *Operating System*, sehingga memungkinkan untuk memonitor, mengontrol dan keperluan otomatisasi berbagai peralatan yang ada didalam rumah ataupun gedung.

Gambar 2. 6 *Home Assistant*

2.7 Algortima Optimisasi *Earthworm*

Earthworm (cacing tanah) merupakan sebuah hewan yang berbentuk *tubeshaped*, binatang bersegmen yang pada umumnya ditemukan di tanah yang menyerap nutrisi dari makhluk hidup yang telah mati. Cacing tanah merupakan makhluk yang mempunyai sistem pencernaan yang kuat yang berada diseluruh tubuhnya. Cacing tanah bernafas dengan menggunakan kulit. Cacing tanah bergerak dengan menggunakan otot yang ada disekujur tubuhnya. Proses perkembangbiakan pada cacing tanah berlangsung pada malam hari. Cacing tanah merupakan hewan *hermaphrodites* karena mempunyai dua kelamin (jantan dan betina) dalam satu tubuh. Lokasi dari organ seksual berada di segmen 9 sampai 15 dengan satu dua pasang testis yang berada dalam sebuah kantung. Sperma yang dikeluarkan dari segmen 15 dihasilkan dan disimpan oleh dua atau empat pasang *vesikula seminalis* yang melalui lubang kelamin jantan. Telur dihasilkan oleh indung telur di segmen 13 sampai dengan lubang kelamin betina yang ada pada segmen 14. Sperma dari cacing lain selama proses pembuahan didapatkan dari kantung yang berada di segmen 9 dan 10. Oleh karena itu, masing-masing cacing, sperma dipindahkan dari segmen 15 ke segmen 9 dan 10. Cacing tanah merupakan hewan yang memiliki kemampuan yang baik dalam hal regenerasi segmen yang hilang/putus tetapi beberapa spesies berbeda memiliki tingkat regenerasi yang berbeda pula. Proses reproduksi dari cacing tanah dapat digunakan untuk optimisasi berbagai macam problem. Tingkah laku cacing tanah saat proses reproduksi dapat diidealkan dengan aturan sebagai berikut [11]:

1. Semua cacing dalam suatu populasi mempunyai kemampuan untuk menghasilkan keturunan dan masing-masing induk hanya mempunyai dua jenis reproduksi.
2. Masing-masing anak cacing yang dihasilkan oleh salah satu jenis reproduksi mempunyai gen yang panjangnya sama dengan induk.
3. Cacing dengan nilai *fitness* terbaik langsung disimpan untuk reproduksi/iterasi selanjutnya dan tidak dapat diubah oleh operator apapun. Hal ini menjamin populasi cacing sekarang selalu lebih baik atau sama dengan populasi sebelumnya.

Cacing tanah memiliki dua jenis reproduksi yaitu sebagai berikut:

- Reproduksi jenis pertama

Cacing tanah merupakan hewan yang mempunyai kelamin jantan dan betina dalam satu tubuh (*hermaphrodites*). Oleh karena itu anak cacing dapat dihasilkan hanya dari satu induk untuk cacing tanah tertentu. Proses reproduksi ini dapat diformulasikan sebagai berikut:

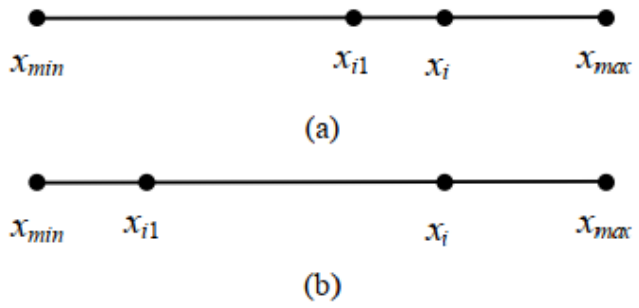
$$X_{i1,j} = X_{max,j} + X_{min,j} - \alpha X_{i,j} \tag{2.1}$$

Dimana $X_{i,j}$ mengindikasikan elemen ke j^{th} dari X_i yang menyajikan posisi dari cacing ke i. Dengan cara yang sama, X_{i1} mengindikasikan elemen ke j^{th} dari X_{i1} yang merupakan posisi baru dari cacing ke i1. $X_{max,j}$ dan $X_{min,j}$ secara berurutan merupakan batas atas dan bawah dari posisi cacing i. $\alpha \in [0,1]$ merupakan faktor kesamaan yang menetukan jarak induk dan anak cacing. Jika α bernilai kecil maka jarak antara induk dan anak pendek, dan X_{i1} berdekatan dengan X_i. Hal ini dapat diimplemetasikan untuk pencarian local antara cacing i seperti pada Gambar 2.7(a). ketika α=0, persamaan (2.1) akan berubah menjadi persamaan (2.2). Cacing i dan cacing generasi baru i memiliki jarak terjauh yaitu $X_{max,j} + X_{min,j}$.

$$X_{i1,j} = X_{max,j} + X_{min,j} \tag{2.2}$$

Sebaliknya jika α bernilai besar maka jarak antara induk dan anak jauh dan x_{i1} berada jauh dari x_i. Hal ini mendorong eksplorasi dan membuat sebuah implementasi metode pencarian global dengan jarak yang panjang seperti pada Gambar 2.7(b). Ketika nilai $\alpha = 1$, maka persamaan (2.1) akan berubah menjadi persamaan (2.3). Cacing i dan cacing generasi baru i memiliki jarak terpendek. Pada titik ini, posisi dari cacing i dan cacing generasi baru i simetris dengan memperhatikan pada persamaan $(x_{max,j} + x_{min,j})/2$. Faktanya, persamaan (2.3) pada dasarnya merupakan sebuah metode pembelajaran *oppositional-based* yang secara luas digunakan dalam berbagai metode optimisasi stokastik [11].

$$X_{i1,j} = X_{max,j} + X_{min,j} - x_{i,j} \tag{2.3}$$



Gambar 2. 7 Gambaran reproduksi dari (a) small α (b) big α [11]

- Reproduksi jenis kedua

Cacing tertentu mampu menghasilkan lebih dari satu anak pada waktu yang sama yang merupakan jenis reproduksi special dari cacing. Reproduksi jenis kedua dapat digunakan untuk berbagai masalah optimasi dengan menambahkan operator *crossover*. M merupakan jumlah anak cacing yang digunakan untuk menghasilkan cacing baru terakhir untuk reproduksi jenis kedua. Secara teori, M dapat berupa sembarang nilai *integer* yang tidak lebih kecil dari nol, namun secara fakta, M bernilai 1,2 dan 3 dalam banyak kasus. N merupakan jumlah induk cacing yang bernilai sembarang *integer* dan bernilai lebih dari satu. N=2, M=1, pada kasus ini, 2 induk cacing dipilih dengan menggunakan *roulette wheel selection*, dan induk hasil dari seleksi diberi nama P_1 dan P_2. Keturunan yang dapat diperoleh dari induk P_1 dan P_2 yaitu:

$$P = \begin{matrix} P_1 \\ P_2 \end{matrix} \tag{2.4}$$

Dengan menggunakan *Uniform crossover*, masing-masing elemen dari dua keturunan x_{12} dan x_{22} dapat dihasilkan seperti pada persamaan (2.5) saat angka acak *rand* lebih besar dari 0.5.

$$\begin{aligned} X_{12,j} &= P_{1,j} \\ X_{22,j} &= P_{2,j} \end{aligned} \tag{2.5}$$

Dimana $x_{12,j}$ dan $x_{22,j}$ merupakan elemen ke j^{th} dari keturunan x_{12} dan x_{22}. Dengan cara yang sama, $P_{1,j}$ dan $P_{2,j}$ merupakan elemen ke j^{th} dari induk P_1 dan P_2. Sebaliknya, keturunan dan induk diperbarui seperti pada persamaan (2.6).

$$\begin{aligned} X_{12,j} &= P_{2,j} \\ X_{22,j} &= P_{1,j} \end{aligned} \tag{2.6}$$

Dan kemudian, cacing hasil reproduksi jenis kedua yaitu x_{i2} dapat ditentukan melalui persamaan (2.7).

$$X_{i2} = \begin{cases} X_{12} & rand < 0.5 \\ X_{22} & else \end{cases} \tag{2.7}$$

Alhasil, dua keturunan x_{12} dan x_{22} dihasilkan oleh *uniform crossover* dengan nilai M=1 dapat dibentuk sebuah algoritma 1 seperti pada Gambar 2.8.

Algorithm 1 Uniform crossover with $M = 1$

Begin
  for $j = 1$ to D (all the elements in an earthworm individual)
  do
    if *rand* > 0.5 **then**
      Generate the j^{th} element of the offsprings x_{12} and x_{22} as equation (12).
    else
      Generate the j^{th} element of the offsprings x_{12} and x_{22} as equation (13).
    end if
  end for j
  Determine the generated earthworm x_{i2} for Reproduction 2 by equation (9).
End.

Gambar 2. 8 Algoritma 1 dengan M=1 [11]

2.8 Algortima Optimisasi *Cuckoo*

Cuckoo search (CS) adalah algoritma optimasi yang dikembangkan oleh Xin-she Yang dan Suash Deb pada tahun 2009. Pencarian *Cuckoo* adalah salah satu dari banyak algoritma yang diilhami alam yang digunakan secara ekstensif untuk mengatasi masalah optimasi di berbagai bidang Teknik [5].

Burung Kukuk (*Cuckoo*) merupakan jenis burung yang menarik, bukan hanya karena suaranya yang merdu tetapi karena strategi reproduksi mereka yang agresif. Beberapa spesies seperti burung Kukuk (*Cuckoo*) Ani dan Guira bertelur di sarang komunal (bersama) meskipun mereka dapat menghapus telur lain untuk meningkatkan kemungkinan penetasan telur mereka sendiri. Cukup banyak spesies burung Kukuk (*Cuckoo*) menggunakan pengeraman parasit obligat dengan cara meletakkan telur mereka ke dalam sarang burung tuan rumah. Terdapat tiga tipe dasar dari pengeraman parasit antara lain pengeraman parasit intraspesifik, pengeraman koopereatif, dan pengambil alihan sarang.

Beberapa burung tuan rumah dapat terlibat konflik langsung dengan burung Kukuk (*Cuckoo*) pengganggu. Jika burung tuan rumah mengetahui bahwa telur tersebut bukan miliknya, mereka akan

melemparkan telur asing tersebut atau meninggalkan telur tersebut dan membangun sarang ditempat baru. Selain itu, waktu bertelur beberapa spesies burung Kukuk juga menakjubkan. Burung Kukuk (*Cuckoo*) parasit seringkali memilih sarang dimana burung tuan rumah hanya meletakkan telurnya sendiri. Secara umum, telur burung Kukuk (*Cuckoo*) menetas sedikit lebih awal daripada telur burung tuan rumah. Setelah burung Kukuk (*Cuckoo*) pertama menetas, tindakan naluri pertama yang akan dilakukannya adalah mengusir telur burung tuan rumah dengan mendorong telur keluar dari sarang. Hal ini meningkatkan jumlah makanan yang telah disediakan oleh burung tuan rumah (*host*) bagi anak burung Kukuk (*Cuckoo*). Studi juga telah menunjukkan bahwa anak burung Kukuk (*Cuckoo*) dapat menirukan suara anak burung tuan rumah (*host*) untuk meningkatkan kesempatan mendapatkan makanan. *Cuckoo search* didasarkan pada tiga aturan ideal [5]:

1. Setiap *Cuckoo* meletakkan satu telur pada satu waktu, dan membuang telurnya di sarang yang dipilih secara acak
2. Sarang terbaik dengan kualitas telur yang tinggi akan terbawa ke generasi berikutnya
3. Jumlah sarang burung tuan rumah yang tersedia bersifat tetap. probabilitas telur yang telah diletakkan oleh burung Kukuk dan ditemukan oleh burung tuan rumah dijabarkan dengan pa ϵ [0,1]. Dalam kasus ini burung tuan rumah dapat membuang telur tersebut atau meninggalkan sarang dan membangun sarang di tempat yang baru

Dalam masalah maksimisasi, kualitas atau kesesuaian dari solusi hanya dapat sebanding dengan nilai fungsi tujuan. Bentuk lain dari kesesuaian dapat didefinisikan dengan cara yang sama dengan fungsi *fitness* dalam algoritma genetika. Dalam pelaksanaan, kita dapat menggunakan representasi sederhana berikut bahwa setiap telur di sarang merupakan solusi, dan masing-masing burung Kukuk (*Cuckoo*) dapat meletakkan hanya satu telur (mewakili satu solusi), tujuannya adalah untuk menggunakan solusi yang baru dan lebih berpotensi untuk menggantikan solusi yang tidak begitu baik dalam sarang. Untuk saat ini, kita akan menggunakan pendekatan yang paling sederhana dimana setiap sarang hanya memiliki satu sel telur. Dalam hal ini, tidak ada perbedaan antara telur, sarang atau burung Kukuk (*Cuckoo*), karena setiap sarang sesuai dengan satu telur yang juga menggambarkan satu burung Kukuk. Berdasarkan tiga aturan tersebut langkah dasar dari *Cuckoo Search* (CS) dapat diringkas sebagai berikut [3][5]:

$$xi\,(t+1) \;=\; xi\,(t) \;+\; \alpha \;\oplus\; L\acute{}evy\,(\lambda) \qquad (2.8)$$

Dimana $xi^{(t+1)}$ adalah generasi solusi baru, i adalah burung kukuk ke-*I,* $\alpha >$ 0 adalah ukuran langkah yang seharusnya berhubungan dengan skala kepentingan masalah tersebut. Dalam kebanyakan kasus dapat digunakan $\alpha = O(L/10)$, L adalah karakteristik skala kepentingan masalah. Sedangkan *Lévy*(λ) menyatakan fungsi persamaan posisi dari *Lévy Flights*, yang bentuk persamaannya adalah sebagai berikut.

$$L´evy \sim u = t^{-\lambda}, (1 < \lambda \leq 3) \quad (2.9)$$

Persamaan 1 merupakan persamaan stokastik untuk langkah acak (*random walk*). Secara umum, langkah acak (*random walk*) adalah rantai Markov yang status / lokasi berikutnya hanya tergantung pada lokasi saat ini dan probabilitas transisi. Lambang ⊕ berarti perkalian entrywise. *Pseudocode* dari *Cuckoo Search Algorithm* (CSA) dapat dijabarkan sebagai berikut:

Algoritma 1 Algoritma Cuckoo Search (Yang dan Deb, 2009).

1: Fungsi Objektif $f(x)$, $x = (x_1 , \dots , x_d)^T$;
2: Inisialisasi populasi dari n sarang burung target X_i (i = 1 , 2 , …, n);
3: **While** (t < generasiTotal) atau (kriteria lain untuk berhenti);
4: Evaluasi nilai kualitas dari masing–masing burung cuckoo
5: Pilih dari burung cuckoo secara acak dan lakukan random walk
6: **Jika** ($F_i > F_j$)
7: Gantikan burung cuckoo j dengan burung cuckoo i
8: **End If**
9: Reset ulang sarang-sarang dengan kondisi terburuk (P_a)
10: Simpan sarang-sarang yang berhasil lolos
11: Urutkan solusi dan cari yang terbaik
12:**End While**

Gambar 2. 9 *Pseudocode* dari *Cuckoo Search Algorithm* (CSA) [12]

2.9 Algortima Optimisasi *Cuckoo-Earthworm*

Cuckoo-Earthworm adalah algoritma optimasi gabungan antara algoritma optimasi *Cuckoo Search* dan *Earthworm Optimization* dan disebut juga algoritma optimasi *hybrid*. Algortima *hybrid* memakai algoritma *Earthworm* hanya saja ditambah mutasi acak pada populasi terententu yang dipilih secara acak.

BAB III
PERANCANGAN

Pada bab ini dibahas perancangan sistem *smart home* meliputi perancangan konseptual dan fungsional.

3.1 Perancangan Konseptual

Perancangan konseptual dari sistem *smart home* yang digunakan yaitu sebagai berikut:

1. *Variable*
   - Jumlah *Smart Plug* (mempengaruhi jumlah peralatan rumah tangga yang digunakan)
   - Besar daya setiap peralatan rumah tangga (mempengaruhi spesifikasi daya *Smart Plug*)
   - Persentase kenyamanan pengguna (mempengaruhi tingkat penghematan daya dan biaya)
   - Harga listrik (mempengaruhi tingkat penghematan daya dan biaya)
2. Parameter
   - Kapasitas daya *Smart Plug* (mempengaruhi jenis peralatan rumah tangga yang bisa dimonitor dan dikontrol)
3. Batasan
   - Jumlah *Smart Plug* (mempengaruhi jumlah peralatan rumah tangga yang bisa dimonitor dan dikontrol)
   - Kapasitas daya *Smart Plug* (mempengaruhi jenis peralatan rumah tangga yang bisa dimonitor dan dikontrol)
4. Kendala
   - Jumlah *Smart Plug* (mempengaruhi jumlah peralatan rumah tangga yang bisa dimonitor dan dikontrol)
   - Kapasitas daya *Smart Plug* (mempengaruhi jenis peralatan rumah tangga yang bisa dimonitor dan dikontrol)
5. Lingkungan sistem
   - *Input* sistem: harga listrik saat ini dan tingkat penghematan daya
   - *Output* sistem: tingkat penghematan daya

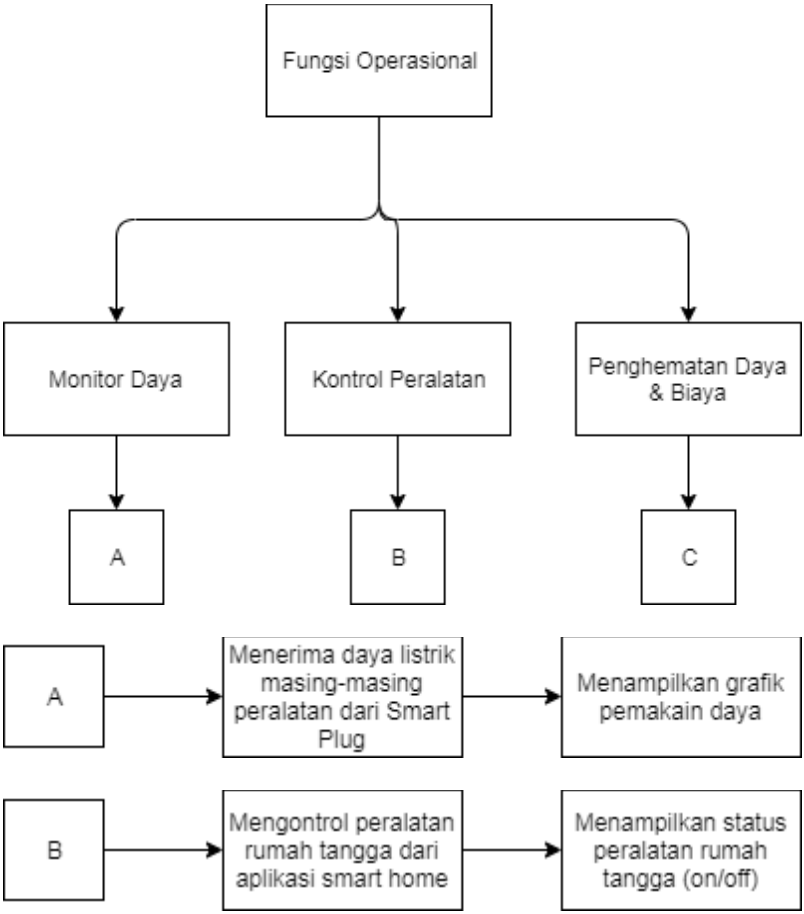
3.2 Perancangan Fungsional

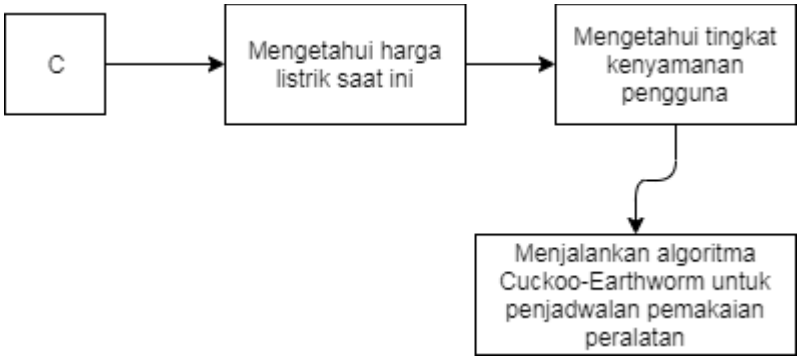
Perancangan fungsional dari sistem *smart home* adalah sebagai berikut:

1. Fungsi operasional
   - Menghemat penggunaan daya listrik
   - Menghemat biaya listrik yang harus dibayarkan
   - Menonitor penggunaan daya listrik

- Mengontrol penggunaan daya listrik secara otomatis

Berikut ini diagram hierarki dari fungsi operasional:





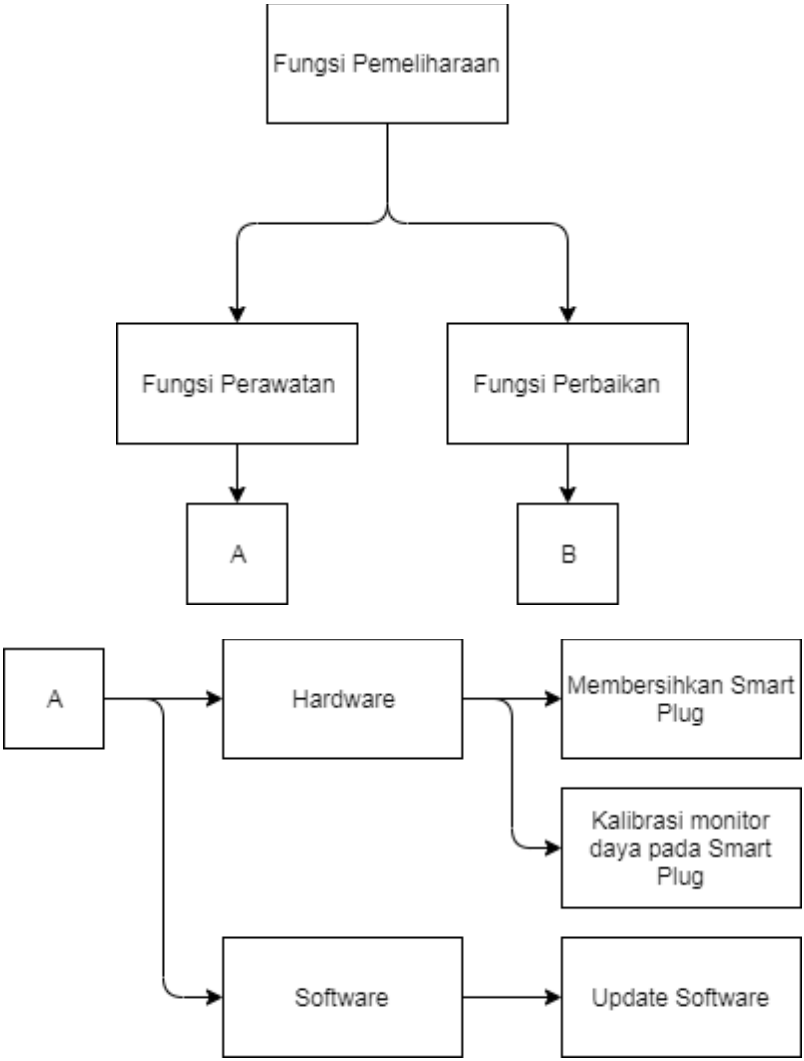
2. Fungsi pemeliharaan

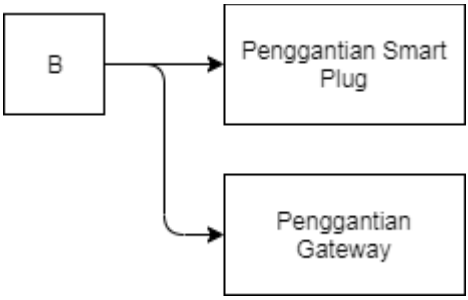
Dalam perancangan sistem *smart home* daya rendah, diinginkan memiliki fungsi-fungsi pemeliharaan sebagai berikut:

a. Fungsi perawatan
   - Membersihkan *Smart Plug* dan *gateway* dari debu
   - Kalibrasi monitor daya pada *Smart Plug*
   - *Update* penyimpanan data pada *gateway*
   - *Update software*

b. Fungsi perbaikan
   - Penggantian *Smart Plug*
   - Penggantian *gateway*

Berikut ini diagram hierarki dari fungsi pemeliharaan:





3.3 Perancangan Detail

Pada bagian ini akan dibahas mengenai perancangan detail yang meliputi *hardware* dan *software* dari sistem *smart home*.

3.3.1 Perancangan *Hardware*

Terdapat beberapa komponen yang digunakan pada sistem *smart home* yaitu Raspberry Pi3, Z-Wave Smart Plug Neo Coolcam, Aeotec Z-Wave USB Stick, Mifi dan 5 buah *interruptible devices*. Raspberry Pi3 digunakan sebagai kontroler, Z-Wave USB Stick sebagai *gateway*, Z-Wave Smart Plug Neo Coolcam digunakan untuk monitor dan kontrol dari berbagai peralatan rumah tangga, Mifi sebagai media komunikasi antara *user* dengan sistem *smart home* dan *interruptible devices* digunakan untuk mewakili beban dari peralatan rumah tangga. Dalam tugas akhir kali ini digunakan 5 buah Neo Coolcam dan 5 *interruptible devices* meliputi *water heater, water pump, air conditioner, refrigerator, iron* dengan spesifikasi daya yang berbeda-beda. Spesifikasi dari Aeotec Z-ZWave USB Stick yang digunakan yaitu:

- USB 2.0 *compliant*
- *Operating distance*: *up to* 500 *feet*/150 *metres outdoors in normal mode or* 1310 *feet*/400 *metres outdoors in PA mode*
- *Radio protocol* Z-Wave
- *Radio Frequency* 908.4 MHz

Spesifikasi dari Smart Plug Z-Wave Neo Coolcam yaitu:

- *Input voltage*: 110-230V AC 50/60Hz
- *Max current*: 13A
- *Radio protocol*: Z-Wave
- *Radio frequency*: 908.4 MHz
- *Wireless distance*: *up to 60m outdoor, up to 40m indoor*

Spesifikasi *interruptible devices* yaitu:

- *Water heater*: 15 kW
- *Water pump*: 9 kW
- *Air conditioner*: 7 kW
- *Refrigerator*: 5 kW
- *Iron*: 3 kW

Tabel 3. 1 Spesifikasi *output*, jumlah konsumsi daya dan *operational time* pada *interruptible devices*

Device	*Output*	Konsumsi Daya	Operational Time (*Hour*)
Water heater	20 ºC – 60 ºC	15 kW – 20 kW	5
Water pump	1 liter/min – 3 liter/min	9 kW – 13 kW	7
Air Conditioner	26 ºC – 22 ºC	7 kW – 9 kW	8
Refrigerator	15 ºC – 0 ºC	5 kW – 8 kW	9
Iron	90 ºC – 120 ºC	3 kW – 7 kW	9

Dalam tugas akhir ini, semakin besar tingkat kenyamanan pengguna maka semakin besar pula daya yang dikonsumsi oleh masing-masing *device*.

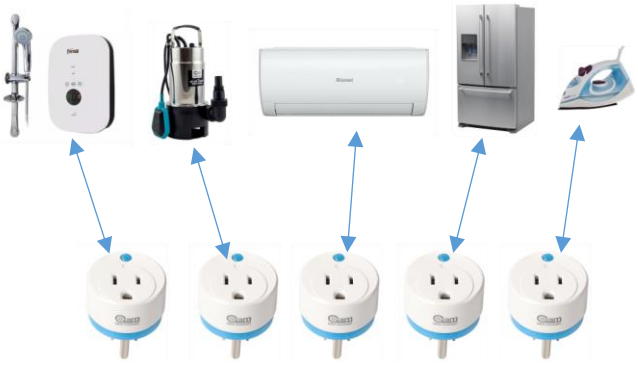
Gambar 3. 1 Raspberry Pi 3 dan Aeotec Z-Wave USB Stick

Gambar 3. 2 *Water heater and Water pump*

Gambar 3. 3 *Air conditioner and Refrigerator*

Gambar 3. 4 *Iron*

Gambar 3. 5 Z-Wave Smart Plug Neo Coolcam



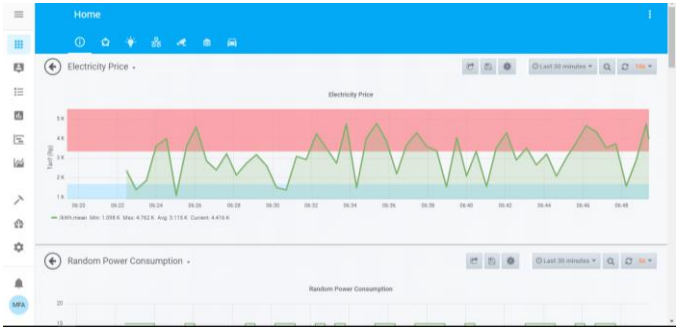
Gambar 3. 6 *Interruptible devices* terhubung dengan *Smart Plug*

3.3.2 Perancangan *User Interface* Aplikasi *Smart Home*

Aplikasi yang digunakan untuk *smart home* yaitu *Home Assistant* yang dibuat dengan menggunakan Bahasa pemrograman Python. Dalam aplikasi *smart home* terdapat beberapa bagian utama yaitu:

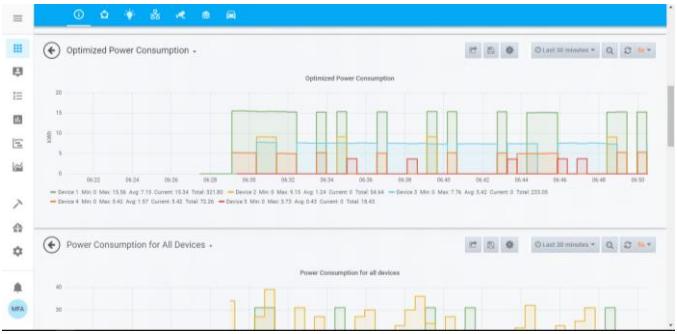
- *Electricity Price*
  Digunakan untuk menampilkan grafik perubahan harga listrik secara *real time* dari waktu ke waktu
- *Random Energy Consumption*
  Digunakan untuk menampilkan grafik konsumsi energi listrik dari waktu ke waktu tanpa adanya optimasi pemakaian energi listrik berdasarkan harga listrik pada masing-masing peralatan
- *Optimized Energy Consumption*
  Digunakan untuk menampilkan grafik konsumsi energi listrik dari waktu ke waktu setelah dilakukan optimasi terhadap pemakaian energi listrik berdasarkan harga listrik dan tingkat kenyamanan pengguna pada masing-masing peralatan
- *Energy Consumption for All Devices*
  Digunakan untuk menampilkan grafik konsumsi energi listrik dari waktu ke waktu untuk seluruh peralatan
- *Total Electricity Cost*
  Digunakan untuk menampilkan grafik tagihan biaya listrik dari waktu ke waktu untuk seluruh peralatan
- Pengaturan untuk algoritma
  Digunakan untuk keperluan *testing* algoritma, untuk input parameter jumlah *device*, populasi, batas bawah dan atas tarif listrik

- *Input* Persentase kenyamanan
  Digunakan untuk memasukkan tingkat kenyamanan pengguna (dalam persen)
- Monitor energi dan biaya sebelum dioptimasi
  Digunakan untuk menampilkan penggunaan energi saat ini, total energi, biaya listrik saat ini dan total biaya listrik sebelum dioptimasi
- Monitor energi dan biaya setelah dioptimasi
  Digunakan untuk menampilkan penggunaan energi saat ini, total energi, biaya listrik saat ini dan total biaya listrik setelah dioptimasi
- Kontrol dan monitor 5 peralatan rumah tangga
  Digunakan untuk monitoring energi dan status (*on*/*off*) pada masing-masing peralatan dan juga sebagai kontrol manual ke peralatan rumah tangga

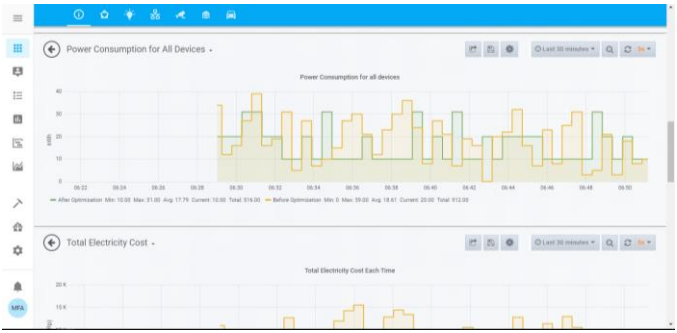


Gambar 3. 7 Bagian *Electricity Price*

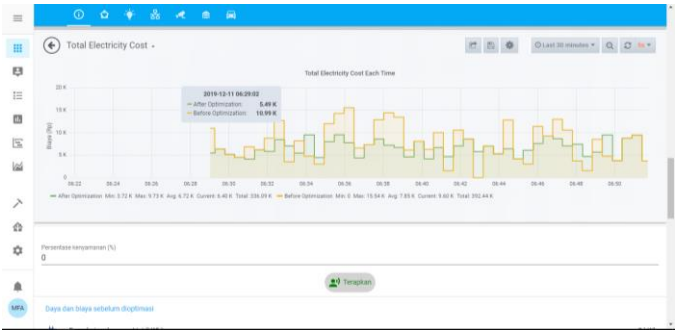
Gambar 3. 8 Bagian *Random Energy Consumption*



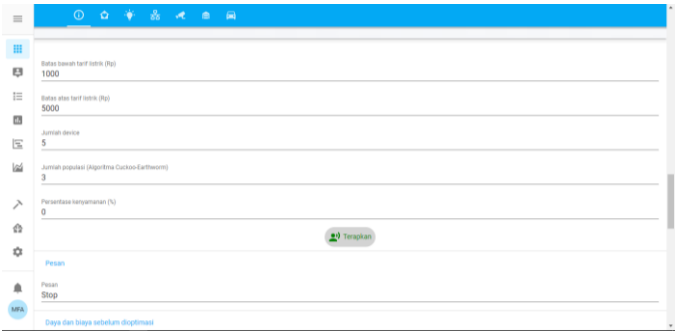
Gambar 3. 9 Bagian *Optimized Energy Consumption*



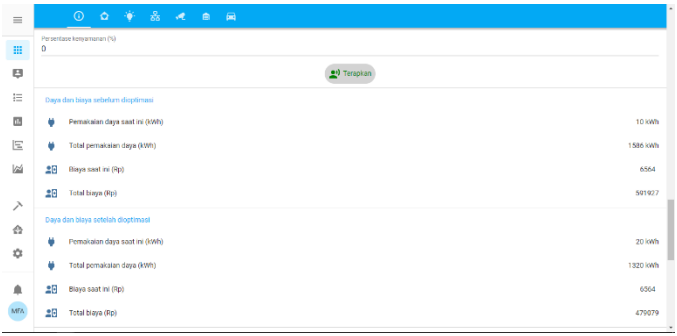
Gambar 3. 10 Bagian *Energy Consumption for All Devices*



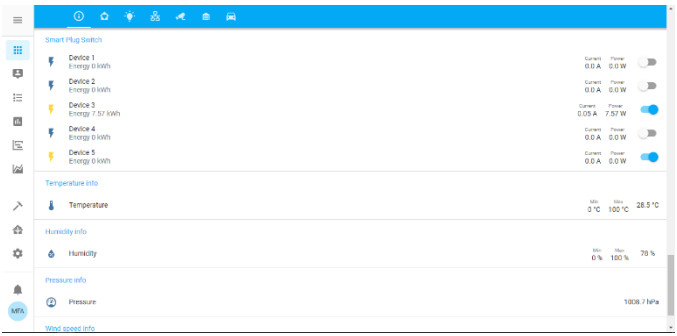
Gambar 3. 11 Bagian *Total Electricity Cost*



Gambar 3. 12 Bagian pengaturan untuk algoritma



Gambar 3. 13 Bagian Input Persentase kenyamanan dan Monitor energi dan biaya sebelum dan sesudah dioptimasi



Gambar 3. 14 Bagian kontrol dan monitor peralatan rumah

3.3.3 Menentukan Harga Listrik secara *Real Time*

Dalam tugas akhir kali ini, harga listrik akan berganti setiap 1 menit (RTP / *Real Time Pricing*) yang mewakili setiap 1 jam dalam kasus yang sebenarnya. Hal ini bertujuan untuk mempercepat pengumpulan data namun tetap bisa diaplikasikan dalam kasus yang sebenarnya. Harga listrik yang digunakan adalah harga listrik berdasarkan daftar besaran tarif tenaga listrik resmi dari Kementerian ESDM dan PLN menurut golongannya.

Tabel 3. 2 Golongan tarif listrik berdasarkan Kementerian ESDM dan PLN

Golongan tarif listrik	Batas daya	Biaya pemakaian
R-1/TR	1.300 VA	Rp 1.467,28/kWh
R-1/TR	2.200 VA	Rp 1.467,28/kWh
R-2/TR	3.500 VA – 5.500 VA	Rp 1.467,28/kWh
R-3/TR	> 6.600 VA	Rp 1.467,28/kWh
B-2/TR	6.600 VA – 200 kVA	Rp 1.467,28/kWh
B-3/TM	> 200 kVA	Rp 1.115/kWh
I-3/TM	> 200 kVA	Rp 1.115/kWh
I-4/TT	> 30.000 kVA	Rp 997/kWh
P-1/TR	6.600 VA – 200 kVA	Rp 1.467/kWh
P-2/TM	> 200 kVA	Rp 1.115/kWh
P-3/TR		Rp 1.467,28/kWh
L/TR, TM, TT		Rp 1.644,52/kWh

Untuk menentukan waktu terjadinya *on-peak* dan *off-peak hours*, maka harga listrik dikategorikan menjadi 3 berdasarkan tabel golongan tarif listrik yaitu:

1. *Low price*: harga listrik Rp. 997/kWh (*off-peak*)
2. *Med price*: harga listrik Rp. 1.115/kWh (*off-peak*)
3. *High price*: harga listrik Rp. 1.467,52/kWh (*on-peak*)

Jadi harga listrik akan berubah secara *random* setiap menit dengan variasi tiga nilai yaitu 997, 1.115, dan 1.467,52.

3.3.4 Formulasi Fungsi Objektif

Berikut ini merupakan fungsi objektif dari sistem *smart home* daya rendah yang digunakan untuk menjalankan fungsi penjadwalan setiap *device* berdasarkan tingkat kenyamanan pengguna dan tarif listrik pada setiap waktu untuk meminimalkan biaya:

$$Fungsi_{objektif} = \sum_{t=1}^{24}\left\{\left(\sum_{i=1}^{n} s_{i(t)} * power_{i(t)} * tarif_{listrik(t)}\right) - threshold\right\} \quad (3.1)$$

Dimana:
$Fungsi_{objektif}$ = Fungsi objektif yang digunakan untuk proses penjadwalan pada setiap device
s_i = Status masing-masing *device* (1/0)
$power_i$ = Daya masing-masing *device* (*Watt*)
$tarif_{listrik}$ = Besar tarif listrik pada setiap waktu (Rp)
Threshold = Biaya *threshold* (Rp)

3.3.5 Menentukan *Power Consumption for each device*

Untuk menentukan besar daya yang dibutuhkan masing-masing *device* berdasarkan tingkat kenyamanan pengguna digunakan persamaan (3.1).

$$pwr = \frac{x-a}{b-a} * (b-a) * (d-c) + c \quad (3.2)$$

Dimana:
pwr = Daya yang dikonsumsi oleh masing *device* berdasarkan tingkat kenyamanan (*Watt*)
x = persentase kenyamanan (%)
a = batas bawah persentase kenyamanan (%)
b = batas atas persentase kenyamanan (%)
c = Daya minimal yang dikonsumsi per *device* (*Watt*)
d = Daya maksimal yang dikonsumsi per *device* (*Watt*)

3.3.6 Menentukan *Random Energy Consumption*

Random Energy Consumption didapat dengan mengalikan 5 buah *rondom binary number* dengan spesifikasi daya pada masing *device*.

$$device_{random_1} = rand_{binary(t)} * power_{1(t)} \quad (3.3)$$
$$device_{random_2} = rand_{binary(t)} * power_{2(t)} \quad (3.4)$$
$$device_{random_3} = rand_{binary(t)} * power_{3(t)} \quad (3.5)$$
$$device_{random_4} = rand_{binary(t)} * power_{4(t)} \quad (3.6)$$
$$device_{random_5} = rand_{binary(t)} * power_{5(t)} \quad (3.7)$$
$$unoptimized_{power} = device_{random1} + device_{random2} + device_{random3} + device_{random4} + device_{random5} \quad (3.8)$$

Total *energy* dalam 1 hari yaitu:

$$total_{energy} = \sum_{t=1}^{24} unoptimized_{energy_i} \quad (3.9)$$

3.3.7 Menentukan *Optimized Energy Consumption*

Optimized Energy Consumption didapat dengan mencari nilai *threshold* yang kemudian dimasukkan ke algoritma *Cuckoo-Earthworm* untuk mendapatkan status dari masing-masing peralatan (*on* atau *off*). Untuk mencari nilai *threshold* dilakukan langkah-langkah sebagai berikut:

- Mencari total biaya dari setiap peralatan

$$device_1 = power_{1(t)} * tarif_{listrik(t)} \quad (3.10)$$
$$device_2 = power_{2(t)} * tarif_{listrik(t)} \quad (3.11)$$
$$device_3 = power_{3(t)} * tarif_{listrik(t)} \quad (3.12)$$
$$device_4 = power_{4(t)} * tarif_{listrik(t)} \quad (3.13)$$
$$device_5 = power_{5(t)} * tarif_{listrik(t)} \quad (3.14)$$
$$total_{biaya} = device_1 + device_2 + device_3 + device_4 + device_5 \quad (3.15)$$

- Menentukan nilai *threshold* dipengaruhi dari persentase alokasi dari harga listrik dan persentase kenyamanan. Untuk tugas akhir kali ini persentase alokasi dari harga listrik dibagi 3 yaitu *Low price* (80%), *Med price* (53.33%) dan *High price* (26.67%). Dan untuk persentase kenyamanan bernilai 0% s/d 100%. Sehingga nilai *threshold* didapat dengan persamaan sebagai berikut:

$$y\% = \frac{x-a}{b-a} * (b-a) * (d-c) + c \quad (3.16)$$

Dimana:
y% = nilai *threshold*
x = persentase kenyamanan (%)
a = batas bawah persentase kenyamanan (%)
b = batas atas persentase kenyamanan (%)
c = batas bawah persentase alokasi harga listrik (%)
d = batas atas persentase alokasi dari harga listrik (%), maka nilai *threshold* yaitu:

$$threshold = y\% * total_{biaya} \quad (3.17)$$

- Menentukan status masing-masing peralatan berdasarkan nilai *threshold* dengan menggunakan algoritma optimisasi *Cuckoo-Earthworm*. Fungsi objektif yang digunakan pada algoritma *Cuckoo-Earthworm* adalah sebagai berikut:

$$status_1 = power_{1(t)} * tarif_{listrik(t)} * s_{1(t)} \quad (3.18)$$
$$status_2 = power_{2(t)} * tarif_{listrik(t)} * s_{2(t)} \quad (3.19)$$
$$status_3 = power_{3(t)} * tarif_{listrik(t)} * s_{3(t)} \quad (3.20)$$
$$status_4 = power_{4(t)} * tarif_{listrik(t)} * s_{4(t)} \quad (3.21)$$
$$status_5 = power_{5(t)} * tarif_{listrik(t)} * s_{5(t)} \quad (3.22)$$
$$f(s_1, s_2, s_3, s_4, s_5) = status_1 + status_2 + status_3 + status_4 + status_5 \quad (3.23)$$
$$f(s_1, s_2, s_3, s_4, s_5) <= threshold \quad (3.24)$$
$$s_1, s_2, s_3, s_4, s_5 = CuckooEarthworm(f(s_1, s_2, s_3, s_4, s_5) = threshold)) \quad (3.25)$$

Dimana:
s_1=status peralatan 1 (0/1)
s_2=status peralatan 2 (0/1)
s_3=status peralatan 3 (0/1)
s_4=status peralatan 4 (0/1)
s_5=status peralatan 5 (0/1)

Sehingga *Optimized Energy Consumption* dapat dicari dengan persamaan sebagai berikut:

$$optimized_{energy} = s_{1(t)} * power_{1(t)} + s_{2(t)} * power_{2(t)} + s_{3(t)} * power_{3(t)} + s_{4(t)} * power_{4(t)} + s_{5(t)} * power_{5(t)} \quad (3.26)$$

Total *energy* dalam 1 hari yaitu:

$$total_{energy} = \sum_{t=1}^{24} optimized_{energy_i} \quad (3.27)$$

3.3.8 Menentukan *Energy Consumption for All devices*

Untuk menamplikan total energi keseluruhan dari peralatan sebelum dan sesudah optimasi pada waktu tertentu, persamaan yang digunakan yaitu persamaan (3.9) dan (3.27).

3.3.9 Menentukan *Total Electricity Cost*

Total electricity Cost dibagi menjadi 2 yaitu Total Electricity Cost sebelum dan sesudah dioptimasi.

- *Total Electricity Cost* sebelum dioptimasi

$$device_{random1} = rand_{binary(t)} * power_{1(t)} * tarif_{listrik(t)} \quad (3.28)$$
$$device_{random2} = rand_{binary(t)} * power_{2(t)} * tarif_{listrik(t)} \quad (3.29)$$
$$device_{random3} = rand_{binary(t)} * power_{3(t)} * tarif_{listrik(t)} \quad (3.30)$$
$$device_{random4} = rand_{binary(t)} * power_{4(t)} * tarif_{listrik(t)} \quad (3.31)$$
$$device_{random5} = rand_{binary(t)} * power_{5(t)} * tarif_{listrik(t)} \quad (3.32)$$

$$unoptimized_{cost} = device_{random1} + device_{random2} + device_{random3} + device_{random4} + device_{random5} \quad (3.33)$$

Total *cost* dalam 1 hari yaitu:

$$total_{cost} = \sum_{t=1}^{24} unoptimized_{cost_i} \quad (3.34)$$

- *Total Electricity Cost* setelah dioptimasi

$$device_1 = s_{1(t)} * power_{1(t)} * tarif_{listrik(t)} \quad (3.35)$$
$$device_2 = s_{2(t)} * power_{2(t)} * tarif_{listrik(t)} \quad (3.36)$$
$$device_3 = s_{3(t)} * power_{3(t)} * tarif_{listrik(t)} \quad (3.37)$$
$$device_4 = s_{4(t)} * power_{4(t)} * tarif_{listrik(t)} \quad (3.38)$$
$$device_5 = s_{5(t)} * power_{5(t)} * tarif_{listrik(t)} \quad (3.39)$$

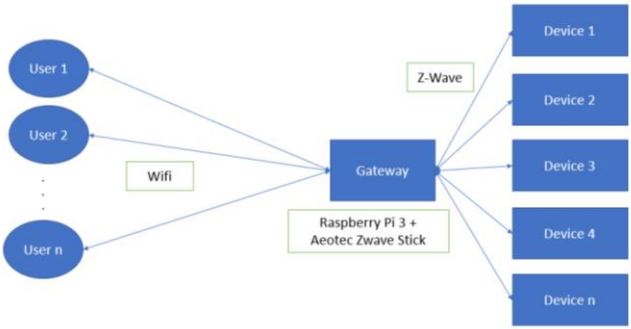
$$optimized_{cost} = device_1 + device_2 + device_3 + device_4 + device_5 \quad (3.40)$$

Total *cost* dalam 1 hari yaitu:

$$total_{cost} = \sum_{t=1}^{24} optimized_{cost_i} \quad (3.41)$$

3.3.10 Arsitektur Sistem *Smart Home*

Semua peralatan rumah tangga terhubung dengan Z-Wave Smart Plug. User dapat mengontrol dan memonitor semua peralatan rumah tangga melalui *smart phone* atau tablet PC yang terhubung dengan *gateway*.

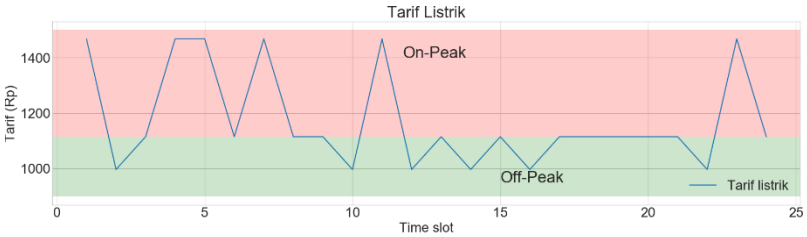


Gambar 3. 15 Sistem keseluruhan *smart home*

User 1, *user* 2 sampai dengan *user* ke-n merupakan pengguna atau penghuni *smart home*. Setiap pengguna dapat mengakses sistem *smart home* menggunakan teknologi *Wi-Fi*. *User* dapat memonitor dan mengontrol setiap peralatan rumah tangga melalui *Smart Phone*, *Laptop* ataupun *Personal Computer*. *User* juga dapat mengontrol tingkat penghematan daya pada setiap peralatan berdasarkan persentase kenyamanan pengguna. *Gateway* berfungsi untuk menghubungkan semua *user* dengan setiap peralatan rumah tangga. Komunikasi antara gateway dengan setiap peralatan rumah tangga (*device*) menggunakan protokol *Z-Wave* yang merupakan protokol komunikasi dengan penggunaan daya yang sangat hemat.

BAB IV
PENGUJIAN DAN PEMBAHASAN

Bab ini membahas hasil simulasi dari sistem *smart home* daya rendah dengan teknik penjadwalan peralatan rumah tangga. Hasil dari optimasi berupa keluaran energi pemakaian dan tarif listrik yang harus dibayar berdasarkan perubahan tarif listrik secara *real time* dan persentase kenyamanan pengguna. Tarif listrik berubah setiap menit dengan nilai *random* berupa tiga variasi nilai yaitu Rp 997, Rp 1.115, dan Rp 1.467,52. Berikut ini merupakan grafik tarif listrik yang berubah setiap 1 menit.



Gambar 4. 1 Grafik perubahan tarif listrik

4.1 Pengujian Terhadap Tingkat Kenyamanan

Berikut ini merupakan pengujian dan analisis hubungan konsumsi energi listrik dan biaya yang harus dibayarkan untuk sebelum dan sesudah dioptimasi terhadap tingkat kenyamanan pengguna (semakin tinggi persentase kenyamanan maka probabilitas peralatan dapat digunakan sewaktu-waktu akan semakin tinggi).

4.1.1 Persentase kenyamanan=0%, jumlah populasi=10

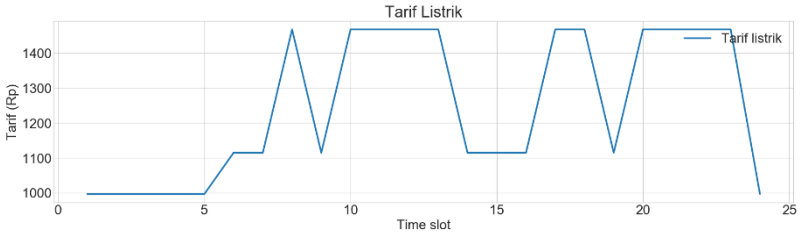
Pemakaian energi listrik dan beban biaya sebelum dilakukan optimasi didapatkan data sebagai berikut:

- Total pemakaian energi: 278 kWh
- Rata-rata pemakaian energi: 11.58 kWh
- Total biaya: Rp. 350,474.27
- Rata-rata biaya: Rp. 14,603.09

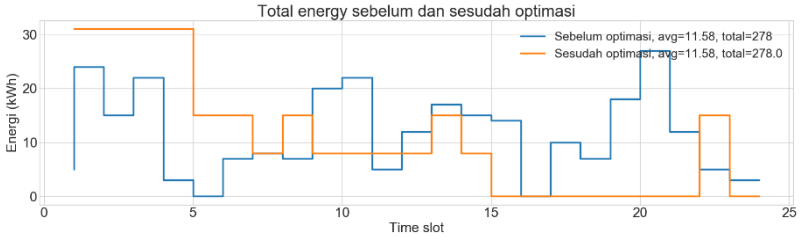
Pemakaian energi listrik dan beban biaya sesudah dilakukan optimasi dengan Teknik penjadwalan peralatan didapatkan data sebagai berikut:

- Total pemakaian energi: 278 kWh
- Rata-rata pemakaian energi: 11.58 kWh
- Total biaya: Rp. 311,068.6
- Rata-rata biaya: Rp. 12,961.19

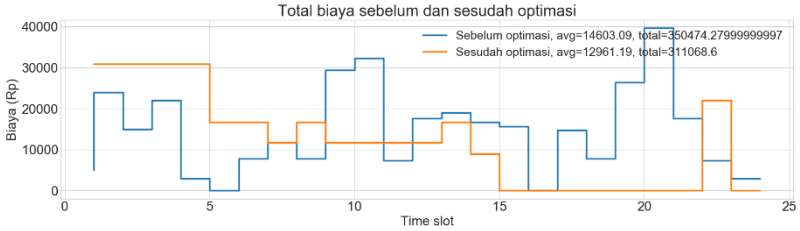
Dari data yang diperoleh terjadi penghematan pada total biaya yaitu sebesar 11.24%.



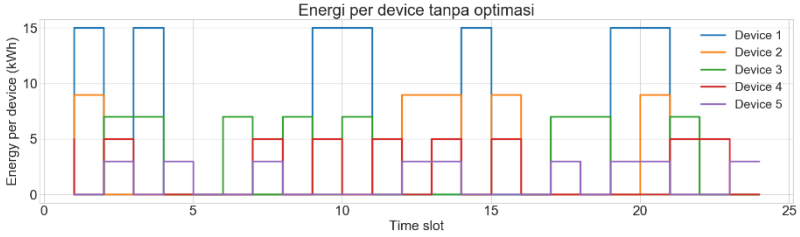
Gambar 4. 2 *Real Time Pricing*



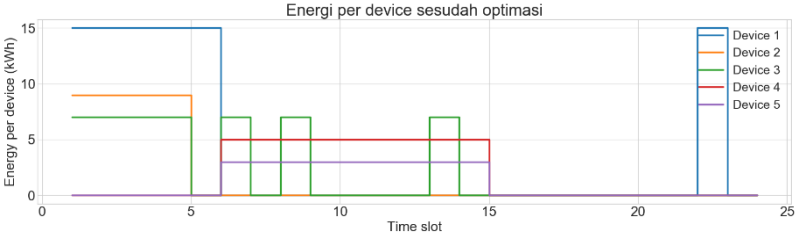
Gambar 4. 3 Pemakaian energi dalam 1 hari



Gambar 4. 4 Total biaya dalam 1 hari



Gambar 4. 5 Energi per *device* sebelum dioptimasi



Gambar 4. 6 Energi per *device* sesudah dioptimasi

4.1.2 Persentase kenyamanan=30%, jumlah populasi=10

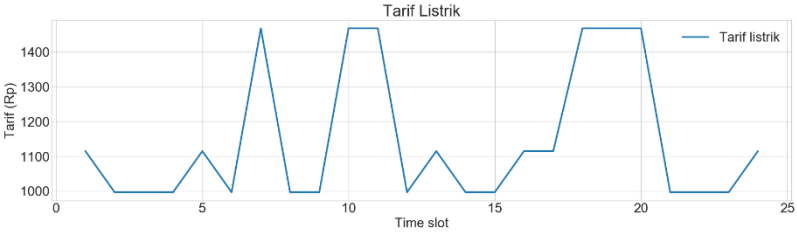
Pemakaian energi listrik dan beban biaya sebelum dilakukan optimasi didapatkan data sebagai berikut:

- Total pemakaian energi: 299 kWh
- Rata-rata pemakaian energi: 12.46 kWh
- Total biaya: Rp. 335,309.6
- Rata-rata biaya: Rp. 13,971.23

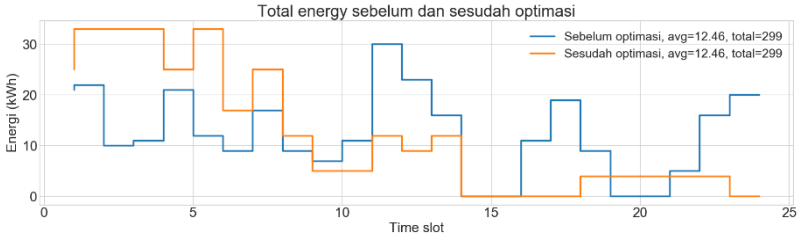
Pemakaian energi listrik dan beban biaya sesudah dilakukan optimasi dengan Teknik penjadwalan peralatan didapatkan data sebagai berikut:

- Total pemakaian energi: 299 kWh
- Rata-rata pemakaian energi: 12.46 kWh
- Total biaya: Rp. 321,533.2
- Rata-rata biaya: Rp. 13,397.22

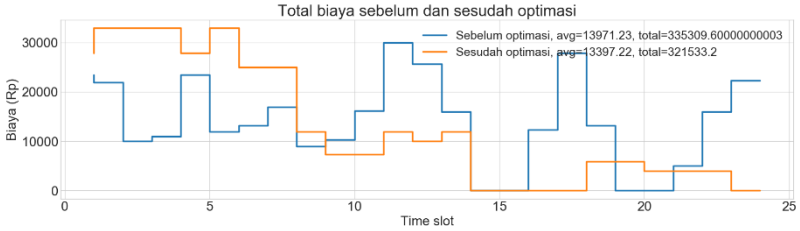
Dari data yang diperoleh terjadi penghematan pada total biaya yaitu sebesar 4.1% dan kenaikan pemakaian total energi sebesar 7% dari tingkat kenyamanan 0%.



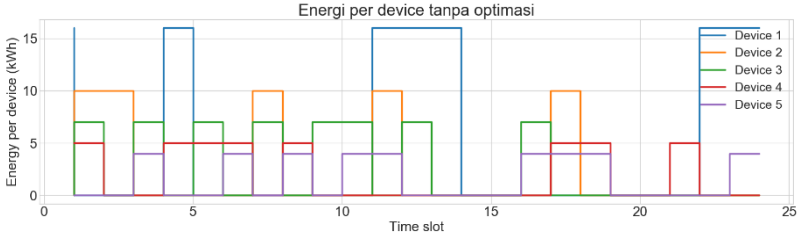
Gambar 4. 7 *Real Time Pricing*



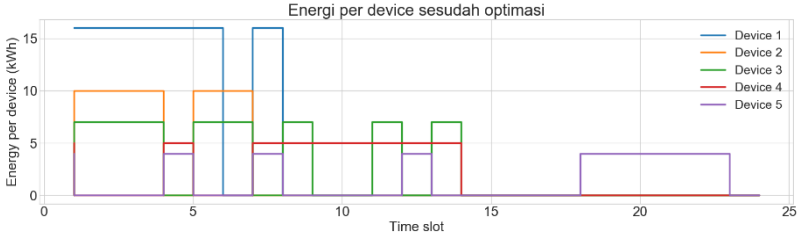
Gambar 4. 8 Pemakaian energi dalam 1 hari



Gambar 4. 9 Total biaya dalam 1 hari



Gambar 4. 10 Energi per *device* sebelum dioptimasi



Gambar 4. 11 Energi per *device* sesudah dioptimasi

4.1.3 Persentase kenyamanan=60%, jumlah populasi=10

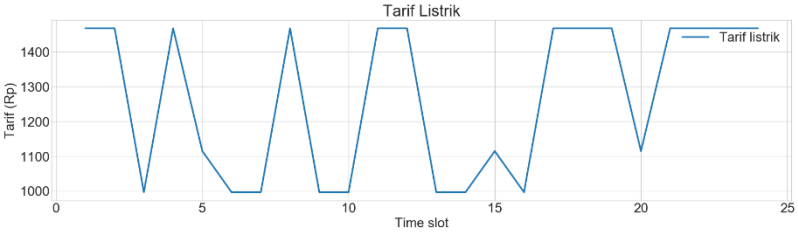
Pemakaian energi listrik dan beban biaya sebelum dilakukan optimasi didapatkan data sebagai berikut:

- Total pemakaian energi: 344 kWh
- Rata-rata pemakaian energi: 14.33 kWh
- Total biaya: Rp. 456,953.32
- Rata-rata biaya: Rp. 19,039.72

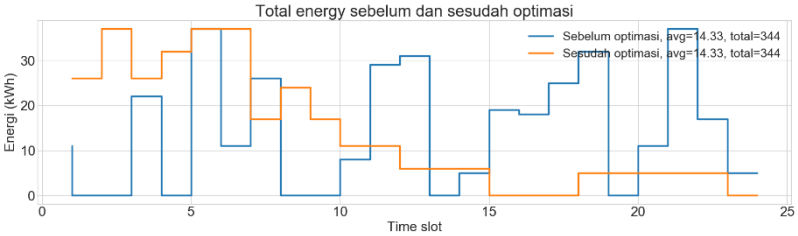
Pemakaian energi listrik dan beban biaya sesudah dilakukan optimasi dengan Teknik penjadwalan peralatan didapatkan data sebagai berikut:

- Total pemakaian energi: 344 kWh
- Rata-rata pemakaian energi: 14.33 kWh
- Total biaya: Rp. 412,503.24
- Rata-rata biaya: Rp. 17,187.63

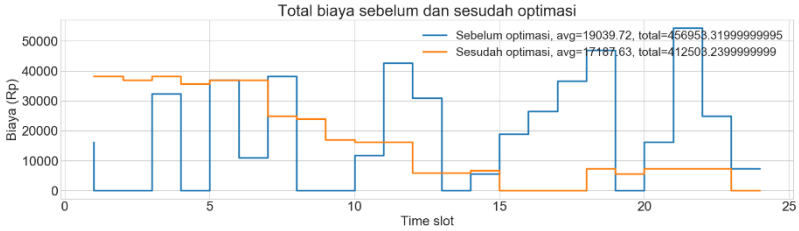
Dari data yang diperoleh terjadi penghematan pada total biaya yaitu sebesar 9.72% dan kenaikan pemakaian total energi sebesar 19.18% dari tingkat kenyamanan 0%.



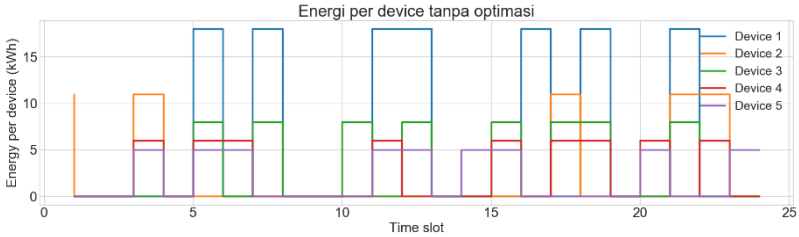
Gambar 4. 12 *Real Time Pricing*



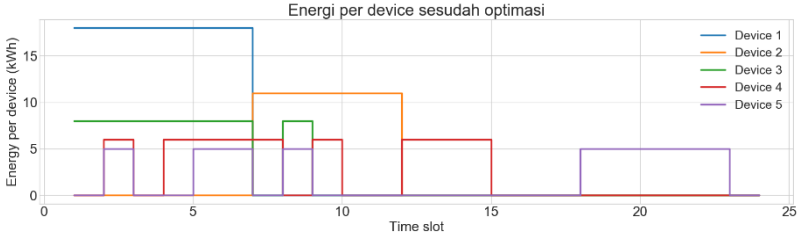
Gambar 4. 13 Pemakaian energi dalam 1 hari



Gambar 4. 14 Total biaya dalam 1 hari



Gambar 4. 15 Energi per *device* sebelum dioptimasi



Gambar 4. 16 Energi per *device* sesudah dioptimasi

4.2 Pengujian Terhadap Jumlah Populasi

Berikut ini merupakan pengujian dan analisis hubungan konsumsi energi listrik dan biaya yang harus dibayarkan untuk sebelum dan sesudah dioptimasi terhadap jumlah populasi pada algoritma *Cuckoo-Earthworm*.

4.2.1 Persentase kenyamanan=0%, jumlah populasi=3

Pemakaian energi listrik dan beban biaya sebelum dilakukan optimasi didapatkan data sebagai berikut:

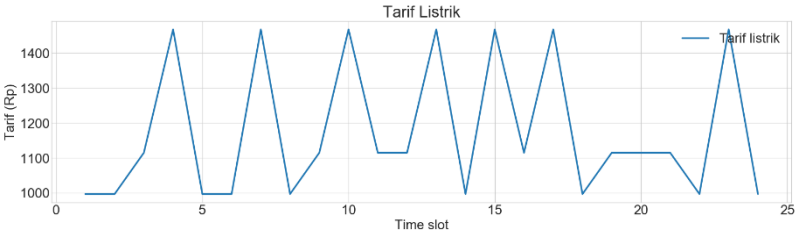
- Total pemakaian energi: 278 kWh
- Rata-rata pemakaian energi: 11.58 kWh

- Total biaya: Rp. 326,616.48
- Rata-rata biaya: Rp. 13,609.02

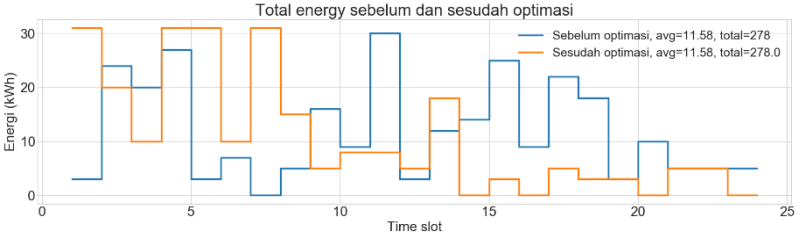
Pemakaian energi listrik dan beban biaya sesudah dilakukan optimasi dengan Teknik penjadwalan peralatan didapatkan data sebagai berikut:

- Total pemakaian energi: 278 kWh
- Rata-rata pemakaian energi: 11.58 kWh
- Total biaya: Rp. 300,714.2
- Rata-rata biaya: Rp. 12,529.76

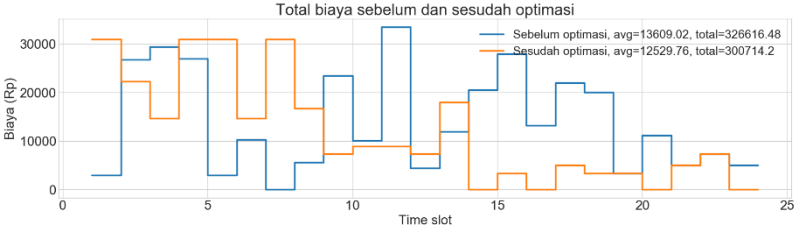
Dari data yang diperoleh terjadi penghematan pada total biaya yaitu sebesar 7.93%.



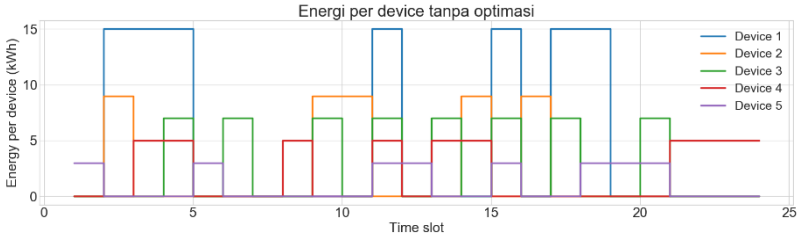
Gambar 4. 17 *Real Time Pricing*



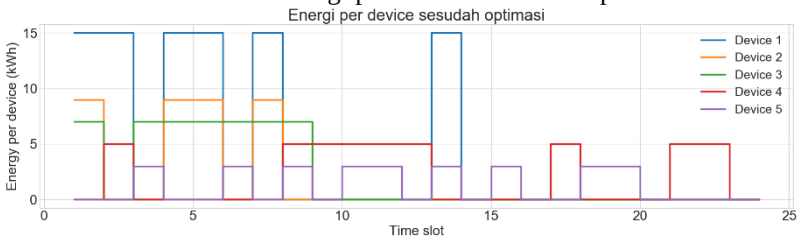
Gambar 4. 18 Pemakaian energi dalam 1 hari



Gambar 4. 19 Total biaya dalam 1 hari



Gambar 4. 20 Energi per *device* sebelum dioptimasi



Gambar 4. 21 Energi per *device* sesudah dioptimasi

4.2.2 Persentase kenyamanan=0%, jumlah populasi=10

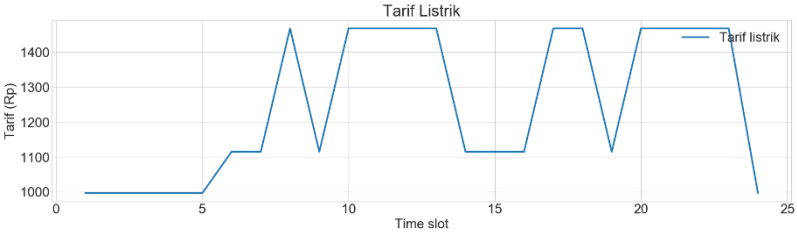
Pemakaian energi listrik dan beban biaya sebelum dilakukan optimasi didapatkan data sebagai berikut:

- Total pemakaian energi: 278 kWh
- Rata-rata pemakaian energi: 11.58 kWh
- Total biaya: Rp. 350,474.27
- Rata-rata biaya: Rp. 14,603.09

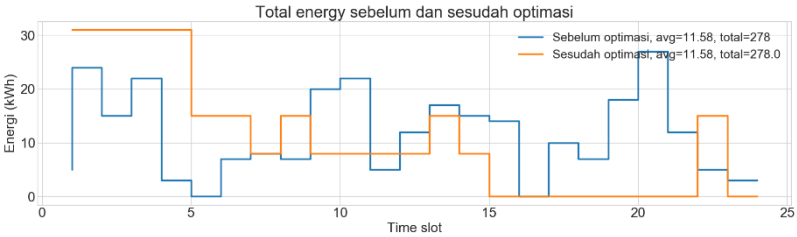
Pemakaian energi listrik dan beban biaya sesudah dilakukan optimasi dengan Teknik penjadwalan peralatan didapatkan data sebagai berikut:

- Total pemakaian energi: 278 kWh
- Rata-rata pemakaian energi: 11.58 kWh
- Total biaya: Rp. 311,068.6
- Rata-rata biaya: Rp. 12,961.19

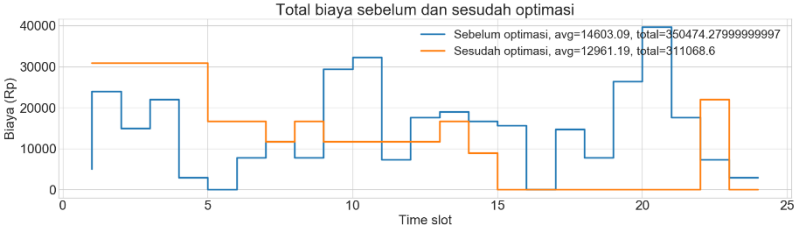
Dari data yang diperoleh terjadi penghematan pada total biaya yaitu sebesar 11.24%.



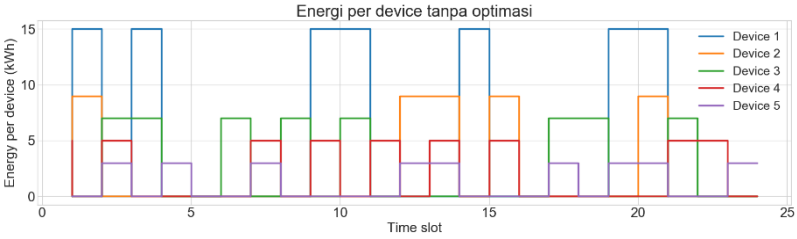
Gambar 4. 22 *Real Time Pricing*



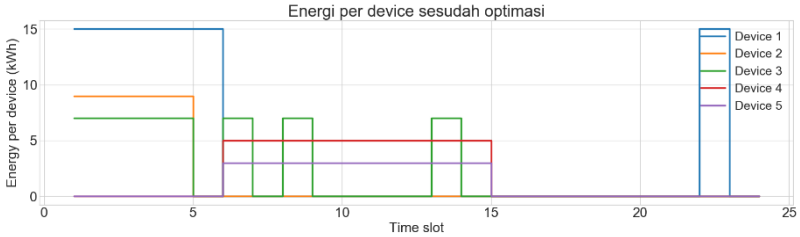
Gambar 4. 23 Pemakaian energi dalam 1 hari



Gambar 4. 24 Total biaya dalam 1 hari



Gambar 4. 25 Energi per *device* sebelum dioptimasi



Gambar 4. 26 Energi per *device* sesudah dioptimasi

4.2.3 Persentase kenyamanan=0%, jumlah populasi=20

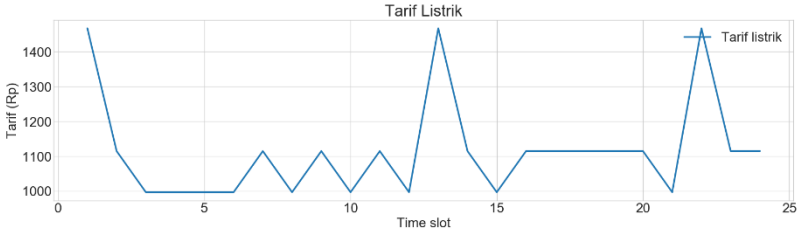
Pemakaian energi listrik dan beban biaya sebelum dilakukan optimasi didapatkan data sebagai berikut:

- Total pemakaian energi: 278 kWh
- Rata-rata pemakaian energi: 11.58 kWh
- Total biaya: Rp. 309,815.0
- Rata-rata biaya: Rp. 12,908.96

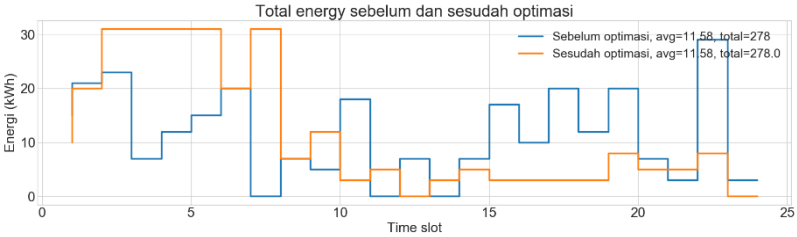
Pemakaian energi listrik dan beban biaya sesudah dilakukan optimasi dengan Teknik penjadwalan peralatan didapatkan data sebagai berikut:

- Total pemakaian energi: 278 kWh
- Rata-rata pemakaian energi: 11.58 kWh
- Total biaya: Rp. 293,781.8
- Rata-rata biaya: Rp. 12,240.91

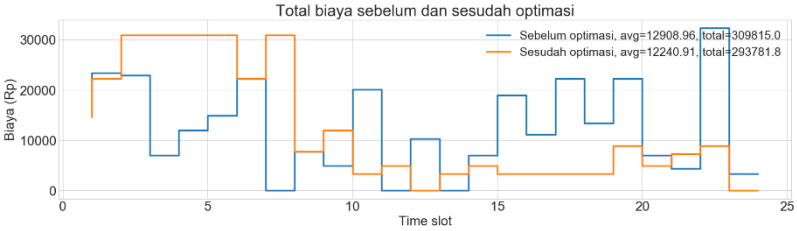
Dari data yang diperoleh terjadi penghematan pada total biaya yaitu sebesar 5.17%.



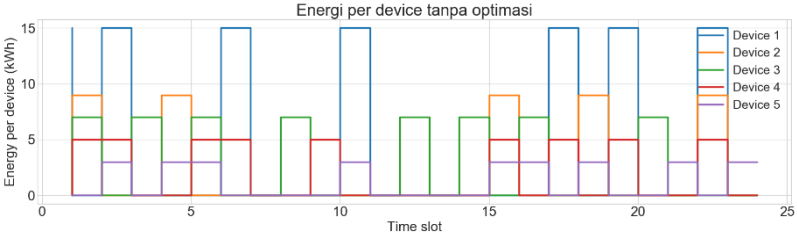
Gambar 4. 27 *Real Time Pricing*



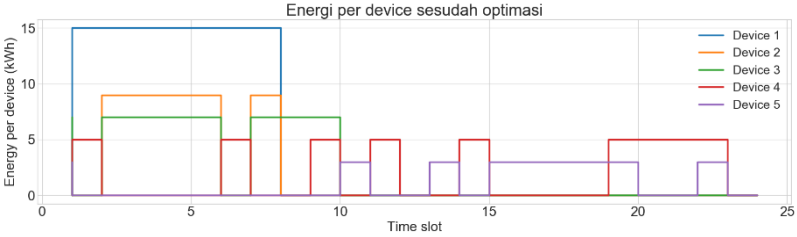
Gambar 4. 28 Pemakaian energi dalam 1 hari



Gambar 4. 29 Total biaya dalam 1 hari



Gambar 4. 30 Energi per *device* sebelum dioptimasi



Gambar 4. 31 Energi per *device* sesudah dioptimasi

4.2.4 Persentase kenyamanan=0%, jumlah populasi=30

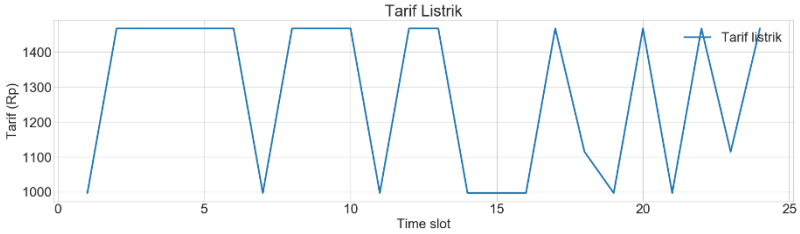
Pemakaian energi listrik dan beban biaya sebelum dilakukan optimasi didapatkan data sebagai berikut:

- Total pemakaian energi: 278 kWh
- Rata-rata pemakaian energi: 11.58 kWh
- Total biaya: Rp. 352,332.67
- Rata-rata biaya: Rp. 14,680.53

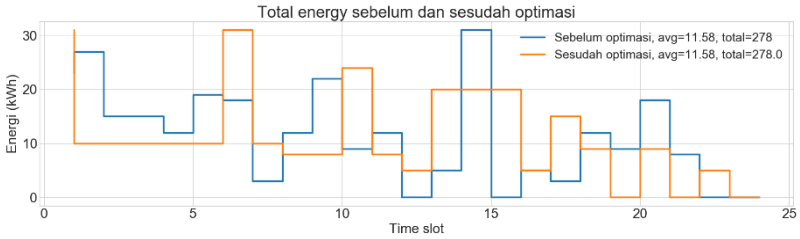
Pemakaian energi listrik dan beban biaya sesudah dilakukan optimasi dengan Teknik penjadwalan peralatan didapatkan data sebagai berikut:

- Total pemakaian energi: 278 kWh
- Rata-rata pemakaian energi: 11.58 kWh
- Total biaya: Rp. 323,754.88
- Rata-rata biaya: Rp. 13,489.79

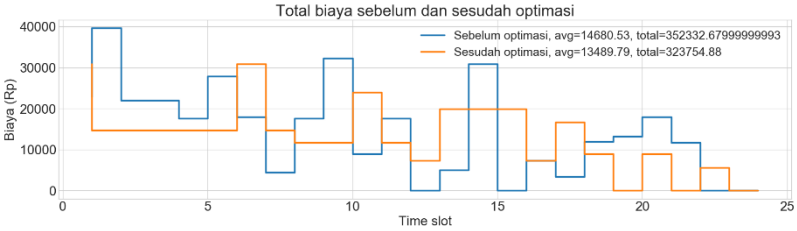
Dari data yang diperoleh terjadi penghematan pada total biaya yaitu sebesar 8.11%.



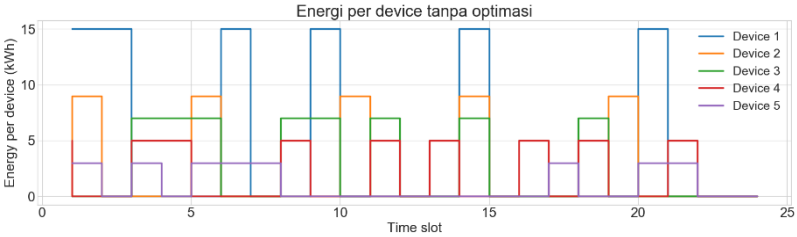
Gambar 4. 32 *Real Time Pricing*



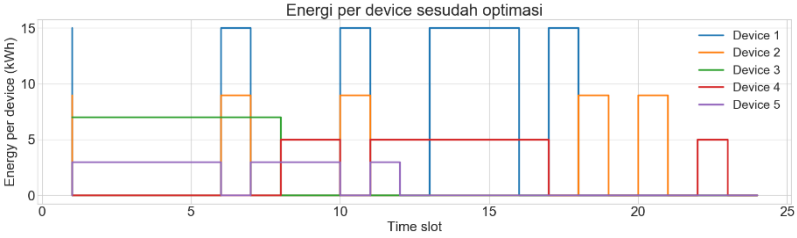
Gambar 4. 33 Pemakaian energi dalam 1 hari



Gambar 4. 34 Total biaya dalam 1 hari



Gambar 4. 35 Energi per *device* sebelum dioptimasi



Gambar 4. 36 Energi per *device* sesudah dioptimasi

4.3 Rangkuman Data Hasil Pengujian

Berikut ini merupakan rangkuman data hasil pengujian dan analisis hubungan konsumsi energi listrik dan biaya yang harus dibayarkan untuk sebelum dan sesudah dioptimasi terhadap persentase kenyamanan dan jumlah populasi pada algortima *Cuckoo-Earthworm*.

Tabel 4.1 merupakan data hasil pengujian hubungan persentase kenyamanan dan jumlah populasi pada algoritma optimisasi *Cuckoo-Earthworm* terhadap penggunaan total energi seluruh *device* dan persentase penghematan total biaya.

Tabel 4. 1 Hubungan persentase kenyaman dan jumlah populasi dengan penggunaan energi dan persentase penghematan biaya

Persentase kenyamanan (%)	Jumlah populasi	Penggunaan total energi (kWh)	penghematan total biaya (%)
0	3	278	7.93
0	10	278	11.24
0	20	278	5.17
0	30	278	8.11
30	3	299	6.4
30	10	299	4.1
30	20	299	9.73
30	30	299	4.92
60	3	344	6.41
60	10	344	9.72
60	20	344	5.97
60	30	344	0.14
100	3	412	5.99
100	10	412	2.91
100	20	412	6.98
100	30	412	1.82

BAB V
KESIMPULAN DAN SARAN

5.1 Kesimpulan

Dari hasil simulasi *smart home* daya rendah dengan optimasi *Cuckoo-Earthworm*, dapat diambil beberapa kesimpulan, diantaranya.

1. Banyaknya konsumsi energi dan biaya tagihan listrik dipengaruhi oleh tingkat kenyamanan pengguna, semakin kecil tingkat kenyamanan pengguna maka energi yang dikonsumsi akan semakin sedikit (tingkat penghematan maksimal).
2. Semakin kecil tingkat kenyamanan pengguna maka persentase penghematan energi hasil optimasi terhadap penggunaan energi pada tingkat kenyamanan maksimal (100%) akan semakin besar.
3. Jumlah populasi pada algoritma *Cuckoo-Earthworm* mempengaruhi persentase penghematan total biaya.
4. Jumlah optimal populasi pada algoritma *Cuckoo-Earthworm* untuk memperbesar persentase penghematan total biaya ada di orde puluhan.
5. Pada tingkat kenyamanan 0% (tingkat kenyamanan paling rendah), tingkat penghematan biaya mencapai 11.24% dan energi mencapai 32.52%.

5.2 Saran

Mempertimbangkan jenis dari peralatan rumah tangga yang dibagi menjadi 3 yaitu *interruptible*, *non-interruptible* dan *base*. Penelitian selanjutnya diharap dapat menambahkan *smart plug* dan juga sumber listrik *alternative*. Mengingat, harga dari *smart plug* dan sumber listrik *alternative* yang relatif mahal. Sehingga kedepannya bisa diterapkan di gedung atau rumah-rumah dengan kebutuhan daya yang kecil sampai besar.

-----Halaman ini sengaja dikosongkan-----

DAFTAR PUSTAKA

[1] Vardakas, J.S., Zorba, N., Verikoukis, C.V.: *A survey on demand response programs in smart grids: pricing methods and optimization algorithms*. IEEE Commun. Surv. Tutor. 17(1), 152–178 (2015).

[2] Jamil Aqib, Nadeem Javaid, and Sheraz Aslam, "*An Efficient Home Energy Optimization by Using Meta-heuristic Techniques While Incorporating Game-theoretic Approach for Real-time Coordination Among Home Appliances*" in 2018 5th Intern. Multi-Topic ICT Conf. (IMTIC), Jamshoro, Pakistan, Apr. 2018.

[3] Jamil A., Javaid N., Khalid M.U., Iqbal M.N., Rashid S., Anwar N. (2019) *An Energy Efficient Scheduling of a Smart Home Based on Optimization Techniques.* In: Barolli L., Xhafa F., Javaid N., Enokido T. (eds) Innovative Mobile and Internet Services in Ubiquitous Computing. IMIS 2018. Advances in Intelligent Systems and Computing, vol 773. Springer, Cham.

[4] Mahmood, A., Javaid, N., Khan, N.A., Razzaq, S.: An optimized approach for home appliances scheduling in smart grid. In: 19th International Multi-topic Conference (INMIC), Islamabad, pp. 1–5 (2016).

[5] Xing B., Gao WJ. (2014) *Cuckoo Inspired Algorithms*. In: Innovative Computational Intelligence: A Rough Guide to 134 Clever Algorithms. Intelligent Systems Reference Library, vol 62. Springer, Cham.

[6] Li L., Xie W., He Z., Xu X., Chen C., Cui X. (2012) *Design of Smart Home Control System Based on ZigBee and Embedded Web Technology*. In: Lei J., Wang F.L., Deng H., Miao D. (eds) Artificial Intelligence and Computational Intelligence. AICI 2012. Lecture Notes in Computer Science, vol 7530. Springer, Berlin, Heidelberg.

[7] Ali M. et al. (2018) *Earth Worm Optimization for Home Energy Management System in Smart Grid*. In: Barolli L., Xhafa F., Conesa J. (eds) Advances on Broad-Band Wireless Computing, Communication and Applications. BWCCA 2017. Lecture Notes on Data Engineering and Communications Technologies, vol 12. Springer, Cham.

[8] Ziming Zhu, Jie Tang, S. Lambotharan, Woon Hau Chin and Zhong Fan, "*An integer linear programming based optimization for home demand-side management in smart grid*," *2012 IEEE PES Innovative Smart Grid Technologies (ISGT)*, Washington, DC, 2012, pp. 1-5.

[9] Eunji Lee and Hyokyung Bahn, "A genetic algorithm based power consumption scheduling in smart grid buildings," *The International Conference on Information Networking 2014 (ICOIN2014)*, Phuket, 2014, pp. 469-474.

[10] A. Agnetis, G. de Pascale, P. Detti and A. Vicino, "*Load Scheduling for Household Energy Consumption Optimization*," in *IEEE Transactions on Smart Grid*, vol. 4, no. 4, pp. 2364-2373, Dec. 2013.

[11] Wang, Gai-Ge & Deb, Suash & Coelho, Leandro. (2015). *Earthworm optimization algorithm: a bio-inspired metaheuristic algorithm for global optimization problems*. International Journal of Bio-Inspired Computation. 10.1504/IJBIC.2015.10004283.

[12] Hardy, Ng Poi Wong & Dedy Suwandi. (2013). *Penerapan Algoritma Cuckoo Search pada Travelling Salesman Problem*. Seminar Nasional Sistem Informasi Indonesia. Medan, 2013.

LAMPIRAN

Program Algoritma Cuckoo-Earthworm

```
import appdaemon.plugins.hass.hassapi as hass
import time
import random
import copy
import math

class Earthworm:
    def __init__(self, dim):
        self.posisi=[0]*dim
        self.nilaiFungsi=0.0

class CuckooEwa(hass.Hass):
    def initialize(self):
        self.log("Hello from AppDaemon")
self.lbl_current_pemakaian_daya=self.args["lbl
_current_pemakaian_daya"]
self.lbl_total_pemakaian_daya=self.args["lbl_t
otal_pemakaian_daya"]
self.lbl_current_pemakaian_daya_bef=self.args[
"lbl_current_pemakaian_daya_bef"]
self.lbl_total_pemakaian_daya_bef=self.args["l
bl_total_pemakaian_daya_bef"]
self.lbl_current_biaya=self.args["lbl_current_
biaya"]
self.lbl_total_biaya=self.args["lbl_total_biay
a"]
self.lbl_current_biaya_bef=self.args["lbl_curr
ent_biaya_bef"]
self.lbl_total_biaya_bef=self.args["lbl_total_
biaya_bef"]
self.lbl_status_operation=self.args["lbl_statu
s_operation"]
        self.device1 = self.args["device1"]
        self.device2 = self.args["device2"]
        self.device3 = self.args["device3"]
        self.device4 = self.args["device4"]
        self.device5 = self.args["device5"]
```

```
        # For Optimized Devices
        self.opt_appl1= self.args['opt_appl1']
        self.opt_appl2= self.args['opt_appl2']
        self.opt_appl3= self.args['opt_appl3']
        self.opt_appl4= self.args['opt_appl4']
        self.opt_appl5= self.args['opt_appl5']
        # For random devices
self.rand_appl1=self.args["random_appl1"]
self.rand_appl2=self.args["random_appl2"]
self.rand_appl3=self.args["random_appl3"]
self.rand_appl4=self.args["random_appl4"]
self.rand_appl5=self.args["random_appl5"]
self.harga_listrik=self.args['harga_listrik']
        self.pb_persen_nyaman =
self.args["pb_persen_nyaman"]
        self.txt_persen_nyaman =
self.args["txt_persen_nyaman"]
        self.txt_jml_device=
self.args["txt_jml_device"]
        self.txt_jml_populasi =
self.args['txt_jml_populasi']
self.txt_off_peak_1=self.args['txt_off_peak_1'
]
self.txt_off_peak_2=self.args['txt_off_peak_2'
]
self.txt_on_peak=self.args['txt_on_peak']
        # Init value ==================
_=self.set_state(self.lbl_current_pemakaian_da
ya_bef, state=0)

_=self.set_state(self.lbl_total_pemakaian_daya
_bef, state=0)

_=self.set_state(self.lbl_current_pemakaian_da
ya, state=0)

_=self.set_state(self.lbl_total_pemakaian_daya
, state=0)
_=self.set_state(self.lbl_current_biaya,
state=0)
        _=self.set_state(self.lbl_total_biaya,
state=0)
```

```
_=self.set_state(self.lbl_current_biaya_bef,
state=0)

_=self.set_state(self.lbl_total_biaya_bef,
state=0)
        # Turn off all lamps
        a=self.turn_off(self.device1)
        a=self.turn_off(self.device2)
        a=self.turn_off(self.device3)
        a=self.turn_off(self.device4)
        a=self.turn_off(self.device5)
        # For operation time each devices
        # Device Random or dev 1 needs 7 hours
to operate in a day
        # Device Random or dev 2 needs 5 hours
to operate in a day
        # Device Random or dev 3 needs 8 hours
to operate in a day
        # Device Random or dev 4 needs 9 hours
to operate in a day
        # Device Random or dev 5 needs 9 hours
to operate in a day
        self.rand_opr_time=[7,5,8,9,9]
        self.opr_time=[7,5,8,9,9]
        self.dev1_price=0
        self.dev2_price=0
        self.dev3_price=0
        self.dev4_price=0
        self.dev5_price=0
        self.thr_price=0
        self.best_pos=0
        # self.persen_nyaman=-1
        self.persen_nyaman=-1
        self.device_power_list=[15, 9, 7, 5,
3]
        self.current_power_bef_opt=0
        self.total_power_bef_opt=0
        self.current_power_aft_opt=0
        self.total_power_aft_opt=0
```

```
        self.current_price_bef_opt=0
        self.total_price_bef_opt=0
        self.current_price_aft_opt=0
        self.total_price_aft_opt=0
        self.off_peak_1=997.0
        self.off_peak_2=1115.0
        self.on_peak=1467.52
        #
==========================================
=======
        # For optimization algorithm
        self.dimensi=5 #x and y #5
        self.minPosisi=0.0 #posisi minimal
dari x and y
        self.maksPosisi=1.0 #posisi maksimal
dari x and y
        self.jumlahIterasi=500 #jumlah iterasi
yang digunakan untuk mencari nilai optimal
fungsi
        # self.ukuranPopulasi=7 #jumlah
populasi keseluruhan
        self.ukuranPopulasi=7
        self.jumlahPopulasiElit=2 #jumlah
populasi elit, populasi elit digunakan untuk
menggantikan populasi dengan nilai terburuk
        self.alpha=0.98 #faktor kemiripan
        self.beta=1 # beta & gamma merupakan
proporsi digunakan untuk menghitung populasi
sebenarnya
        self.gamma=0.9
        self.probCrossover=1 #probabilitas
crossover
        self.probMutasi=0.3 #probabilitas
mutasi
        self.posisiTerbaik=0 #
        # For Smart Home
        self.run()

    def sorter(self, arr):
        n = len(arr)
        for i in range(n):
            for j in range(0, n-i-1):
```

```
                if abs(arr[j].nilaiFungsi) >
abs(arr[j+1].nilaiFungsi) :
                    arr[j].nilaiFungsi,
arr[j+1].nilaiFungsi = arr[j+1].nilaiFungsi,
arr[j].nilaiFungsi
                    arr[j].posisi,
arr[j+1].posisi = arr[j+1].posisi,
arr[j].posisi
        return arr
    #Roulette Wheel Selection
    def RouletteWheelSelection(self, populasi,
rnd):
        bobot=[0.0]*len(populasi)
        totalBobot=0.0
        for i in range(len(bobot)):
            totalBobot +=
1/(populasi[i].nilaiFungsi+0.0001)
            bobot[i]=totalBobot
        probabilitasKumulatif=rnd*totalBobot
        idxTerpilih=-1
        for i in range(len(bobot)):
            if bobot[i]>probabilitasKumulatif:
                idxTerpilih=i
                break
        return idxTerpilih
    def hitungNilaiFungsi(self, posisi):
        hasil=(posisi[0])*self.dev1_price *
round(self.opr_time[0]/(self.opr_time[0]+0.000
1)) + \
            (posisi[1])*self.dev2_price *
round(self.opr_time[1]/(self.opr_time[1]+0.000
1)) + \
            (posisi[2])*self.dev3_price *
round(self.opr_time[2]/(self.opr_time[2]+0.000
1)) + \
            (posisi[3])*self.dev4_price *
round(self.opr_time[3]/(self.opr_time[3]+0.000
1)) + \
            (posisi[4])*self.dev5_price *
round(self.opr_time[4]/(self.opr_time[4]+0.000
1)) - self.thr_price
```

```
        return hasil

    def EAO(self):
        print('Start finding, please wait...')
        posisiTerbaik=[0.0]*self.dimensi
#inisialisasi posisi terbaik (x & y)
        nilaiFungsiTerbaik=0.0 #init nilai
fungsi terbaik
        best_res=self.thr_price
        best_pos=0
        res=0
        pos=0
        populasi=[Earthworm(self.dimensi) for
i in range(self.ukuranPopulasi)] #init
earthworm sejumlah populasi
        #memberi posisi acak pada masing-
masing EW, dan memastikan bahwa tidak ada
posisi yang sama antara EW1 dg yg lainnya
        for i in range(self.ukuranPopulasi):
            isUnik=False
            tmpPosisi=[0.0]*self.dimensi
            while not isUnik:
                isUnik=True
                for k in range(self.dimensi):

tmpPosisi[k]=(self.maksPosisi-
self.minPosisi)*[random.random() for _ in
range(0, 1)][0]+self.minPosisi
                for j in range(i):
                    for jd in
range(self.dimensi):
                        if
populasi[j].posisi[jd]==tmpPosisi[jd]:
                            isUnik=False
                        else:
                            isUnik=True
                            break
                    if not isUnik: break
            for k in range(self.dimensi):

populasi[i].posisi[k]=tmpPosisi[k]
```

```
            # hitung nilai fungsi pada posisi
tersebut

populasi[i].nilaiFungsi=self.hitungNilaiFungsi
(populasi[i].posisi)
        # Sorting for the best function value
        populasi=self.sorter(populasi)
        # find the temporary best position and
function value
        posisiTerbaik=populasi[0].posisi

nilaiFungsiTerbaik=populasi[0].nilaiFungsi
        # start the iteration
        iterasi=0
        wait='.'
        while iterasi<self.jumlahIterasi:
            iterasi +=1
            wait += '.'

populasiElit=[Earthworm(self.dimensi) for i in
range(self.jumlahPopulasiElit)]
            for i in
range(self.jumlahPopulasiElit):

populasiElit[i]=copy.deepcopy(populasi[i])
            self.beta *= self.gamma

populasiSebelumnya=[Earthworm(self.dimensi)
for i in range(self.ukuranPopulasi)]
            totalPosisi=[0.0]*self.dimensi
            for i in
range(self.ukuranPopulasi):

populasiSebelumnya[i]=copy.deepcopy(populasi[i
])
                for j in range(self.dimensi):
                    totalPosisi[j] +=
populasiSebelumnya[i].posisi[j]

#perhitungan reproduksi jenis pertama
```

```
populasiPertama=[Earthworm(self.dimensi) for i
in range(self.ukuranPopulasi)]
            for i in
range(self.ukuranPopulasi):
                for j in range(self.dimensi):
populasiPertama[i].posisi[j]=self.maksPosisi+s
elf.minPosisi-self.alpha*populasi[i].posisi[j]
                #perhitungan reproduksi jenis
kedua
                tmpPosisi=[0.0]*self.dimensi
                #tentukan nilai indeks acak
dalam populasi

idxPopulasiAcak=round((self.ukuranPopulasi-
1)*[random.random() for _ in range(0,
1)][0]+0.5)
                #jika indeks populasi yang
sedang dihitung masih kurang dari jumlah
populasi elit,
                #maka nilai populasi sementara
diambil dari populasi dg indeks acak yg telah
ditentukan
                if i<self.jumlahPopulasiElit:
                    for j in
range(self.dimensi):

tmpPosisi[j]=populasi[int(idxPopulasiAcak)].po
sisi[j]
                else:
                    #tentukan indeks acak
kedua untuk dijadikan sebagai induk kedua,
                    #oleh sebab itu, nilai
indeks acak kedua tdk boleh sama dg indeks
acak pertama.
                    #Teknik yang digunakan
adalah teknik seleksi Roulette Wheel

idxPopulasiAcak2=idxPopulasiAcak
                    m=0
                    while
idxPopulasiAcak2==idxPopulasiAcak:
                        m=m+1
```

```
idxPopulasiAcak2=self.RouletteWheelSelection(p
opulasi, [random.random() for _ in range(0,
1)][0])
                        if idxPopulasiAcak2==-
1:
                            idxPopulasiAcak2=1
                        if m>=3:
                            break
                    #catat induk pertama dan
kedua yang telah didapatkan dari posisi acak

induk1=copy.deepcopy(populasi[int(idxPopulasiA
cak)])

induk2=copy.deepcopy(populasi[idxPopulasiAcak2
])
                    #Lakukan proses crossover
antara posisi dari induk pertama dan kedua
                    #Catat hasil crossover
sebagai hasil populasi sementara yang
ditemukan
                    for j in
range(self.dimensi):

nilaiAcak=[random.random() for _ in range(0,
1)][0]
                        if
self.probCrossover>nilaiAcak:

tmpPosisi[j]=induk1.posisi[j] if nilaiAcak>0.5
else induk2.posisi[j]
                        else:

tmpPosisi[j]=induk2.posisi[j] if nilaiAcak>0.5
else induk1.posisi[j]
                #hitung posisi sebenarnya dari
masing-masing EW dengan formula:
                #posisi
baru=beta*populasi1+(1-beta)*posisi sementara
```

```
                #kemudian hitung nilai fungsi
pada posisi tsb
                for j in range(self.dimensi):

populasi[i].posisi[j]=self.beta*populasiPertam
a[i].posisi[j]+(1-self.beta)*tmpPosisi[j]

populasi[i].nilaiFungsi=self.hitungNilaiFungsi
(populasi[i].posisi)

            #lakukan pengurutan populasi bds
nilai fungsi tertinggi ke nilai fungsi
terendah
            #pengurutan ini dimaksudkan agar
populasi elit tidak mengalami mutasi
            populasi=self.sorter(populasi)
            #berikut adalah proses mutasi
dengan menggunakan metode mutasi Cauchy

bobotCauchy=copy.deepcopy(posisiTerbaik)
            #lakukan proses mutasi pada semua
populasi selain populasi elit
            for i in
range(self.jumlahPopulasiElit,
self.ukuranPopulasi, 1):
                #tentukan nilai acak antara 0
- 1
                #jika nilai acak kurang dari
probabilitas mutasi, maka hitung bobot pada
masing-masing dimensi dg rumus:
                #bobot=total posisi dimensi /
ukuran populasi
                for j in range(self.dimensi):
                    if
self.probMutasi>[random.random() for _ in
range(0, 1)][0]:

bobotCauchy[j]=totalPosisi[j]/self.ukuranPopul
asi
                #hitung nilai posisi baru
dengan menggunakan rumus:
```

```
                #posisi baru = (bobot+posisi
terbaik)/2
                for j in range(self.dimensi):

populasi[i].posisi[j]=(bobotCauchy[j]+posisiTe
rbaik[j])/2
            #lakukan perulangan pada masing-
masing EW
            #jika posisi EW berada diluar
batas posisi yang diperbolehkan,
            #maka kembalikan nilainya agar
masuk kedalam batas yg diperbolehkan
            #kemudian hitung nilai fungsi pada
posisi tersebut
            for i in
range(self.ukuranPopulasi):
                for j in range(self.dimensi):
                    if
populasi[i].posisi[j]<self.minPosisi:

populasi[i].posisi[j]=self.minPosisi
                    elif
populasi[i].posisi[j]>self.maksPosisi:

populasi[i].posisi[j]=self.maksPosisi

populasi[i].nilaiFungsi=self.hitungNilaiFungsi
(populasi[i].posisi)

            #jika terdapat EW pada posisi yang
sama, maka EW dengan indeks yang lebih tinggi-
            #akan diganti posisinya dengan
posisi acak.
            #kemudian dilakukan pengecekan
lagi agar tidak ada EW pada posisi yang sama
            for i in
range(self.jumlahPopulasiElit):
                populasi[len(populasi)-1-
i]=copy.deepcopy(populasiElit[i])
            for i in
range(self.ukuranPopulasi):
```

```
                isUnik=False
                while not isUnik:
                    isUnik=True
                    idxTerakhir=-1
                    for j in range(i):
                        for jd in
range(self.dimensi):
                            if
populasi[i].posisi[jd]==populasi[j].posisi[jd]
:
                                isUnik=False
                            else:
                                isUnik=True
                                break
                        if not isUnik:
                            idxTerakhir=i
                            break
                    if not isUnik:
                        for jd in
range(self.dimensi):

populasi[idxTerakhir].posisi[jd]=(self.maksPos
isi-self.minPosisi)*[random.random() for _ in
range(0, 1)][0]+self.minPosisi
            #proses pengurutan populasi pada
posisi yang baru
            populasi=self.sorter(populasi)
            #jika nilai fungsi EW terbaik
ternyata lebih baik dari nilai fungsi secara
umum,
            #maka ambil posisi terbaik EW tsb
sebagai posisi terbaik
            if abs(populasi[0].nilaiFungsi) <
abs(nilaiFungsiTerbaik):

posisiTerbaik=populasi[0].posisi

nilaiFungsiTerbaik=populasi[0].nilaiFungsi
                # Find the best solution

pos=[round(populasi[0].posisi[0]),
round(populasi[0].posisi[1]),
```

```
round(populasi[0].posisi[2]),
round(populasi[0].posisi[3]),
round(populasi[0].posisi[4])]

res=self.hitungNilaiFungsi(pos)
                if(abs(res) < abs(best_res))
and (res<=0):
                    best_res=res
                    best_pos=pos

            # Find the best solution
            pos=[round(populasi[0].posisi[0]),
round(populasi[0].posisi[1]),
round(populasi[0].posisi[2]),
round(populasi[0].posisi[3]),
round(populasi[0].posisi[4])]
            res=self.hitungNilaiFungsi(pos)
            if(abs(res) < abs(best_res)) and
(res<=0):
                best_res=res
                best_pos=pos
            # Mutation for posisition
            #mutation rate = pm
            pm = 0.15
            no_of_mutations = int(round(pm *
10))#int(np.round(pm * 10))
            # Calculating the Generation
number
            # gen_num = np.random.randint(0,
10 - 1, no_of_mutations)
            gen_num = [random.randint(0, 10 -
1) for i in range(no_of_mutations)]

gen=populasi[0].posisi+populasi[1].posisi

gen[gen_num[0]]=random.random()#np.random.rand
()
            # Mutation posisition for elite
population
            populasi[0].posisi=gen[0:5]
            populasi[1].posisi=gen[5:]
```

```
        if best_pos==0:
            best_pos=[0]*self.dimensi
        self.best_pos=best_pos

    def pb_persen_nyaman_state(self):

state=self.get_state(self.pb_persen_nyaman)
        if state=='on':
            status =
self.set_state(self.pb_persen_nyaman, state =
"off")
            self.persen_nyaman =
int(self.get_state(self.txt_persen_nyaman))
            self.dimensi=
int(self.get_state(self.txt_jml_device))
            self.ukuranPopulasi=
int(self.get_state(self.txt_jml_populasi))
self.off_peak_1=float(self.get_state(self.txt_
off_peak_1))
self.off_peak_2=float(self.get_state(self.txt_
off_peak_2))
self.on_peak=float(self.get_state(self.txt_on_
peak))
            if int(self.dimensi)<=0 or
self.off_peak_1<0 or self.off_peak_2<0 or
self.on_peak<0 or self.on_peak<self.off_peak_2
or self.off_peak_2<self.off_peak_1 or
self.ukuranPopulasi<2:
                self.persen_nyaman=-1
_=self.set_state(self.lbl_status_operation,
state="Invalid input !")
            else:
_=self.set_state(self.lbl_status_operation,
state="Penerapan sukses.")

    def run(self):
_=self.set_state(self.lbl_status_operation,
state="Start")
        self.log("Start")
        harga_listrik_buff=0.0
        n=1
        pwr_bef=[]
```

```
        pwr=[]
        cost_bef=[]
        cost=[]
        price_=[]

        opt_dev1=[]
        opt_dev2=[]
        opt_dev3=[]
        opt_dev4=[]
        opt_dev5=[]

        # Wait until start button in clicked
        while (self.persen_nyaman==-1):
            # get pb persen nyaman state and
persen nyaman
            self.pb_persen_nyaman_state()
            time.sleep(0.2)
        # Calculate power consumption of each
device based on comfort level
self.device_power_list[0]=math.floor((float(se
lf.persen_nyaman)-0)/(100-0)*(20-15)+15)
self.device_power_list[1]=math.floor((float(se
lf.persen_nyaman)-0)/(100-0)*(13-9)+9)
self.device_power_list[2]=math.floor((float(se
lf.persen_nyaman)-0)/(100-0)*(9-7)+7)
self.device_power_list[3]=math.floor((float(se
lf.persen_nyaman)-0)/(100-0)*(8-5)+5)
self.device_power_list[4]=math.floor((float(se
lf.persen_nyaman)-0)/(100-0)*(7-3)+3)
        # Power for random devices
        pos=random.sample(range(1, 25), 24)
        dev1_state=[self.device_power_list[0]
if n<=self.rand_opr_time[0] else 0 for n in
pos]
        pos=random.sample(range(1, 25), 24)
        dev2_state=[self.device_power_list[1]
if n<=self.rand_opr_time[1] else 0 for n in
pos]
        pos=random.sample(range(1, 25), 24)
```

```
        dev3_state=[self.device_power_list[2]
if n<=self.rand_opr_time[2] else 0 for n in
pos]
        pos=random.sample(range(1, 25), 24)
        dev4_state=[self.device_power_list[3]
if n<=self.rand_opr_time[3] else 0 for n in
pos]
        pos=random.sample(range(1, 25), 24)
        dev5_state=[self.device_power_list[4]
if n<=self.rand_opr_time[4] else 0 for n in
pos]

        while n<=24:
            # get harga listrik
harga_listrik=float(self.get_state(self.harga_
listrik))
            harga_listrik_buff=harga_listrik

price_.append(float(harga_listrik))

            # get total harga of each device
(kWh)

self.dev1_price=self.device_power_list[0]*floa
t(harga_listrik)*round(self.opr_time[0]/(self.
opr_time[0]+0.0001))
self.dev2_price=self.device_power_list[1]*floa
t(harga_listrik)*round(self.opr_time[1]/(self.
opr_time[1]+0.0001))
self.dev3_price=self.device_power_list[2]*floa
t(harga_listrik)*round(self.opr_time[2]/(self.
opr_time[2]+0.0001))
self.dev4_price=self.device_power_list[3]*floa
t(harga_listrik)*round(self.opr_time[3]/(self.
opr_time[3]+0.0001))
self.dev5_price=self.device_power_list[4]*floa
t(harga_listrik)*round(self.opr_time[4]/(self.
opr_time[4]+0.0001))
total_harga=self.dev1_price+self.dev2_price+se
lf.dev3_price+self.dev4_price+self.dev5_price
            # get price total for scheduling
            # y=(x-a)/(b-a)*(d-c)+c
```

```
            # get current price status and
determine the persentase of price for
scheduling
price_status=self.get_state(self.harga_listrik
, attribute="price_status")
            persen_status=0
            if str(price_status)=="low":
                persen_status=80
            elif str(price_status)=="medium":
                persen_status=53.33
            else: #high
                persen_status=26.67
            # Calculate persentase for
scheduling
            y=(float(self.persen_nyaman)-
0)/(100-0)*(80-persen_status)+ persen_status
            # Calculate Threshold price for
current elect price and user comfort
            self.thr_price=(y/100)*total_harga

            self.EAO()
            pos=self.best_pos

            # Find Current Power BEFORE
optimization
appl=[0]*len(self.device_power_list)
            appl[0]=dev1_state[n-1]
            appl[1]=dev2_state[n-1]
            appl[2]=dev3_state[n-1]
            appl[3]=dev4_state[n-1]
            appl[4]=dev5_state[n-1]

            _=self.set_state(self.rand_appl1,
state=int(appl[0]))
            _=self.set_state(self.rand_appl2,
state=int(appl[1]))
            _=self.set_state(self.rand_appl3,
state=int(appl[2]))
            _=self.set_state(self.rand_appl4,
state=int(appl[3]))
```

```
            _=self.set_state(self.rand_appl5,
state=int(appl[4]))

            current_pwr=[]
            if 24-n==self.opr_time[0] and 24-
n>0:

opt_dev1.append(self.device_power_list[0])

self.opr_time[0]=self.opr_time[0]-1
            else:

opt_dev1.append([(self.device_power_list[0]*in
t(round(self.opr_time[0]/(self.opr_time[0]+0.0
001)))) if pos[0]==1 else 0][0])
                if int(pos[0])>0 and
self.opr_time[0]>0:

self.opr_time[0]=self.opr_time[0]-1

            if 24-n==self.opr_time[1] and 24-
n>0:

opt_dev2.append(self.device_power_list[1])

self.opr_time[1]=self.opr_time[1]-1
            else:

opt_dev2.append([(self.device_power_list[1]*in
t(round(self.opr_time[1]/(self.opr_time[1]+0.0
001)))) if pos[1]==1 else 0][0])
                if int(pos[1])>0 and
self.opr_time[1]>0:

self.opr_time[1]=self.opr_time[1]-1

            if 24-n==self.opr_time[2] and 24-
n>0:

opt_dev3.append(self.device_power_list[2])

self.opr_time[2]=self.opr_time[2]-1
```

```
            else:

opt_dev3.append([(self.device_power_list[2]*in
t(round(self.opr_time[2]/(self.opr_time[2]+0.0
001)))) if pos[2]==1 else 0][0])
                if int(pos[2])>0 and
self.opr_time[2]>0:

self.opr_time[2]=self.opr_time[2]-1

            if 24-n==self.opr_time[3] and 24-
n>0:

opt_dev4.append(self.device_power_list[3])

self.opr_time[3]=self.opr_time[3]-1
            else:

opt_dev4.append([(self.device_power_list[3]*in
t(round(self.opr_time[3]/(self.opr_time[3]+0.0
001)))) if pos[3]==1 else 0][0])
                if int(pos[3])>0 and
self.opr_time[3]>0:

self.opr_time[3]=self.opr_time[3]-1

            if 24-n==self.opr_time[4] and 24-
n>0:

opt_dev5.append(self.device_power_list[4])

self.opr_time[4]=self.opr_time[4]-1
            else:
opt_dev5.append([(self.device_power_list[4]*in
t(round(self.opr_time[4]/(self.opr_time[4]+0.0
001)))) if pos[4]==1 else 0][0])
                if int(pos[4])>0 and
self.opr_time[4]>0:
self.opr_time[4]=self.opr_time[4]-1
            # Set state for optimized device
that is used for graph
```

```
            _=self.set_state(self.opt_appl1,
state=int(opt_dev1[n-1]))
            _=self.set_state(self.opt_appl2,
state=int(opt_dev2[n-1]))
            _=self.set_state(self.opt_appl3,
state=int(opt_dev3[n-1]))
            _=self.set_state(self.opt_appl4,
state=int(opt_dev4[n-1]))
            _=self.set_state(self.opt_appl5,
state=int(opt_dev5[n-1]))
            # Set actual device (on/off)
            # Set state of each devices
            a=[self.turn_on(self.device1) if
opt_dev1[n-1]>0 else
self.turn_off(self.device1)]
            a=[self.turn_on(self.device2) if
opt_dev2[n-1]>0 else
self.turn_off(self.device2)]
            a=[self.turn_on(self.device3) if
opt_dev3[n-1]>0 else
self.turn_off(self.device3)]
            a=[self.turn_on(self.device4) if
opt_dev4[n-1]>0 else
self.turn_off(self.device4)]
            a=[self.turn_on(self.device5) if
opt_dev5[n-1]>0 else
self.turn_off(self.device5)]
            # For calculation
            current_pwr.append(opt_dev1[n-1])
            current_pwr.append(opt_dev2[n-1])
            current_pwr.append(opt_dev3[n-1])
            current_pwr.append(opt_dev4[n-1])
            current_pwr.append(opt_dev5[n-1])
            # Find Current Power BEFORE Opt
self.current_power_bef_opt=int(appl[0])+int(ap
pl[1])+int(appl[2])+int(appl[3])+int(appl[4])
_=self.set_state(self.lbl_current_pemakaian_da
ya_bef, state=int(self.current_power_bef_opt))

pwr_bef.append(self.current_power_bef_opt)
            # Find Total Power BEFORE
optimization
```

```
self.total_power_bef_opt+=self.current_power_b
ef_opt
_=self.set_state(self.lbl_total_pemakaian_daya
_bef, state=int(self.total_power_bef_opt))
            # Find Current price BEFORE
optimization
self.current_price_bef_opt=sum([float(harga_li
strik)*int(i) if int(i)>0 else 0 for i in
appl])
_=self.set_state(self.lbl_current_biaya_bef,
state=int(self.current_price_bef_opt))

cost_bef.append(self.current_price_bef_opt)
# Find Total price BEFORE optimization
self.total_price_bef_opt+=self.current_price_b
ef_opt
_=self.set_state(self.lbl_total_biaya_bef,
state=int(self.total_price_bef_opt))
            # Find Current Power AFTER
optimization
            self.current_power_aft_opt=sum([i
for i in current_pwr])
_=self.set_state(self.lbl_current_pemakaian_da
ya, state=int(self.current_power_aft_opt))
pwr.append(self.current_power_aft_opt)
            # Find Total Power AFTER
optimization

self.total_power_aft_opt+=self.current_power_a
ft_opt
_=self.set_state(self.lbl_total_pemakaian_daya
, state=int(self.total_power_aft_opt))
            # Find Current price AFTER
optimization
self.current_price_aft_opt=sum([float(harga_li
strik)*i for i in current_pwr])
_=self.set_state(self.lbl_current_biaya,
state=int(self.current_price_aft_opt))
cost.append(self.current_price_aft_opt)
            # Find Total price AFTER
optimization
```

```
self.total_price_aft_opt+=self.current_price_a
ft_opt
_=self.set_state(self.lbl_total_biaya,
state=int(self.total_price_aft_opt))
# Detecting of current price change
            while
harga_listrik==harga_listrik_buff:

harga_listrik=float(self.get_state(self.harga_
listrik))
                # get pb persen nyaman state
and persen nyaman
                #
self.pb_persen_nyaman_state()
                time.sleep(0.2)
            if
harga_listrik!=harga_listrik_buff:
_=self.set_state(self.lbl_status_operation,
state="Iterasi ke "+ str(n))
            n=n+1
_=self.set_state(self.lbl_status_operation,
state="Iterasi ke "+str(n)+", Stop.")
```

BIODATA PENULIS

Miftahul Falahi Alfafa lahir di Tulungagung, 24 November 1995. Penulis menyelesaikan pendidikan dasar di MI Miftahul Huda Kota Tulungagung dari tahun 2002 hingga 2008. Setelah itu, penulis melanjutkan pendidikan di MTSN Karangrejo dan SMKN 3 Boyolangu Kota Tulungagung. Penulis melanjutkan ke jenjang perguruan tinggi di Institut Teknologi Sepuluh Nopember. Pada masa perkuliahan penulis banyak menghabiskan waktu di laboratorium Sistem Pengaturan. Penulis juga aktif dalam kegiatan di luar kampus.

email: miftahf77@gmail.com