TUGAS AKHIR – IF184802

IMPLEMENTASI GENETIC ALGORITHM UNTUK OPTIMASI SLEEP SCHEDULING PADA WIRELESS SENSOR NETWORK

CHENDRA SENA OEMARYOGA
NRP 05111640000063

Dosen Pembimbing I
Waskitho Wibisono, S.Kom., M.Eng., Ph.D.

Dosen Pembimbing II
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

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

TUGAS AKHIR – IF184802

IMPLEMENTASI GENETIC ALGORITHM UNTUK OPTIMASI SLEEP SCHEDULING PADA WIRELEES SENSOR NETWORK

CHENDRA SENA OEMARYOGA
NRP 05111640000063

Dosen Pembimbing I
Waskitho Wibisono, S.Kom., M.Eng., Ph.D.

Dosen Pembimbing II
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

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

(Halaman ini sengaja dikosongkan)

UNDERGRADUATE THESIS – IF184802

IMPLEMENTATION OF GENETIC ALGORITHM FOR OPTIMIZING SLEEP SCHEDULING ON WIRELESS SENSOR NETWORK

CHENDRA SENA OEMARYOGA
NRP 05111640000063

First Advisor
Waskitho Wibisono, S.Kom., M.Eng., Ph.D.

Second Advisor
Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

DEPARTMENT OF INFORMATICS
Faculty of Intelligent Electrical and Informatics Technology
Sepuluh Nopember Institute of Technology
Surabaya 2020

(Halaman ini sengaja dikosongkan)

LEMBAR PENGESAHAN

IMPLEMENTASI GENETIC ALGORITHM UNTUK OPTIMASI SLEEP SCHEDULING PADA WIRELESS SENSOR NETWORK

TUGAS AKHIR

Diajukan untuk Memenuhi Salah Satu Syarat
Memperoleh Gelar Sarjana Komputer
pada
Bidang Studi Komputasi Berbasis Jaringan
Program Studi S-1 Departemen Teknik Informatika
Fakultas Teknologi Elektro dan Informatika Cerdas
Institut Teknologi Sepuluh Nopember

Oleh:
CHENDRA SENA OEMARYOGA
NRP: 05111640000063

Disetujui oleh Pembimbing tugas akhir:

1. Waskitho Wibisono, S.Kom.,M.Eng.,Ph.D. (Pembimbing 1)
   (NIP. 197410222000031001)

2. Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D. - .....
   (NIP. 197708242006041001) (Pembimbing 2)

SURABAYA
JUNI, 2020

(Halaman ini sengaja dikosongkan)

IMPLEMENTASI GENETIC ALGORITHM UNTUK OPTIMASI SLEEP SCHEDULING PADA WIRELESS SENSOR NETWORK

Nama Mahasiswa	**:**	**Chendra Sena Oemaryoga**
NRP	**:**	**05111640000063**
Departemen	**:**	**Teknik Informatika FTEIC ITS**
Dosen Pembimbing 1	**:**	**Waskitho Wibisono, S.Kom., M.Eng., Ph.D.**
Dosen Pembimbing 2	**:**	**Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.**

Abstrak

Wireless sensor network (WSN) adalah arsitektur jaringan yang terdiri dari banyak *node* yang bekerja secara kooperatif memantau lingkungan yang besar. *Wireless sensor network* terbentuk dari sekumpulan *sensor node* yang diletakkan tersebar di suatu area, dan *base station* sebagai pengumpul informasi dari semua *node*.

Node yang berjalan pada lingkungan WSN beroperasi menggunakan baterai, yang artinya energi pada *node* sangat terbatas. Maka perlu dibuat metode agar *node* dapat beroperasi lebih lama meskipun energinya sangat terbatas. Salah satu metode yang bisa digunakan adalah *sleep scheduling*.

Pada tugas akhir ini, sensor node akan disebar pada area berbentuk *grid*, dan dilakukan pemetaan wilayah jangkauan berdasarkan *grid* yang telah dibuat. Dengan begitu dapat dilihat seberapa besar wilayah jangkauan semua *node* dan wilayah mana saja yang tumpang tindih. *Sleep scheduling* dibuat menggunakan optimasi *genetic algorithm*, dengan tujuan mengurangi konsumsi energi, mengurangi jangkauan *node* yang tumpang tindih, namun tidak mempengaruhi jangkauan *node* secara keseluruhan.

Kata kunci: ***Wireless Sensor Network, Sleep Scheduling, Genetic Algorithm*****.**

IMPLEMENTATION OF GENETIC ALGORITHM FOR OPTIMIZING SLEEP SCHEDULING ON WIRELESS SENSOR NETWORK

Student's Name : Chendra Sena Oemaryoga
Student's ID : 05111640000063
Department : Informatics FTEIC-ITS
First Advisor : Waskitho Wibisono, S.Kom., M.Eng., Ph.D.
Second Advisor : Royyana Muslim Ijtihadie, S.Kom., M.Kom., Ph.D.

Abstract

Wireless Sensor Network (WSN) is a network artchitecture consisting of many nodes that work cooperatively to monitor large environment. Wireless sensor network consist of sensor nodes that are placed scattered in an area, and the base station as a collector of information from all nodes.

Nodes that run in the WSN environment operate using batteries, which means that energy at the node is very limited. Then the method needs to be made so that the node can operate longer even though the energy is very limited. One method that can be used is sleep scheduling.

In this final project, sensor nodes will be deployed on a grid-shaped area, and a mapping of the coverage area based on the grid has been made. That way it can be seen how much the coverage area of all nodes and any overlapping region. Sleep scheduling is optimized using genetic algorithm, with the aim of reducing energy consumption, reducing the range of overlapping nodes, but does not affect the overall node range.

Keyword: Wireless Sensor Network, Sleep Scheduling, Genetic Algorithm.

(Halaman ini sengaja dikosongkan)

KATA PENGANTAR

بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Puji syukur kepada Allah Yang Maha Esa atas segala karunia dan rahmat-Nya sehingga penulis dapat menyelesaikan tugas akhir yang berjudul

"IMPLEMENTASI GENETIC ALGORITHM UNTUK OPTIMASI SLEEP SCHEDULING PADA WIRELESS SENSOR NETWORK".

Semoga apa yang tertulis di dalam buku tugas akhir ini dapat bermanfaat bagi pengembangan ilmu pengetahuan saat ini dan ke depannya, serta dapat memberikan kontribusi yang nyata.

Dalam pelaksanaan dan pembuatan tugas akhir ini tentunya sangat banyak bantuan yang penulis terima dari berbagai pihak, tanpa mengurangi rasa hormat penulis ingin mengucapkan terima kasih sebesar-besarnya kepada:

1. Allah SWT. Dan Nabi Muhammad SAW. yang telah membimbing penulis selama hidup.
2. Keluarga penulis yang selalu memberikan bantuan berupa waktu, materi, dan semangat pada penulis.
3. Bapak Waskitho Wibisono, S.Kom., M.Eng.,Ph.D. dan Bapak Royyana Muslim Ijtihadie, S.Kom.,M.Kom., Ph.D selaku Dosen Pembimbing penulis yang telah membimbing, memberikan nasihat, dan memotivasi penulis sehingga penulis dapat menyelesaikan tugas akhir ini.
4. Ibu Dr. Eng. Chastine Fatichah, S.Kom., M.Kom. selaku kepala Departemen Teknik Informatika ITS.
5. Bapak dan Ibu Dosen yang telah memberikan ilmunya selama penulis berkuliah di Teknik Informatika ITS.
6. Teman-teman Angkatan 2016 yang sudah menemani penulis melalui masa-masa awal kuliah hingga sekarang.
7. Teman-teman admin lab LP2 (Jonathan, Lita, Rizaldi) yang selalu memberi semangat dan bantuan, bahkan setelah perkuliahan diliburkan karena pandemi.

8. Teman-teman satu komunitas yang sering memberikan support dan hiburan.
9. Untuk orang-orang yang tidak dapat disebutkan satu persatu oleh penulis dan pembaca buku tugas akhir ini.

Penulis telah berusaha sebaik-baiknya dalam menyusun tugas akhir ini. Namun, penulis memohon maaf apabila terdapat kekurangan, kesalahan maupun kelalaian yang telah penulis lakukan. Kritik dan saran yang membangun dapat disampaikan sebagai bahan perbaikan selanjutnya. Tetap semangat dalam menjalani kehidupan, jangan menyerah, karena Allah masih ingin melihat kita berjuang. Semoga kita semua selalu diberi kebahagiaan lahir dan batin dan kesuksesan dunia akhirat. Aamiin.

Surabaya, 23 Juni 2020

Chendra Sena Oemaryoga

DAFTAR ISI

LEMBAR PENGESAHAN ..... **v**
Abstrak ..... **vii**
Abstract ..... **ix**
KATA PENGANTAR ..... **xi**
DAFTAR ISI ..... **xiii**
DAFTAR GAMBAR ..... **xvii**
DAFTAR TABEL ..... **xix**
KODE SUMBER ..... **xxi**
BAB I PENDAHULUAN **1**
1.1 Latar Belakang 1
1.2 Rumusan Masalah 2
1.3 Batasan Permasalahan 2
1.4 Tujuan 2
1.5 Manfaat 3
1.6 Metodologi 3
1.6.1 Penyusunan Proposal Tugas Akhir 3
1.6.2 Studi Literatur 3
1.6.3 Implementasi Sistem 4
1.6.4 Pengujian dan Evaluasi 4
1.6.5 Penyusunan Buku 4
1.7 Sistematika Penulisan Laporan 4
BAB II TINJAUAN PUSTAKA **7**
2.1 *Wireless Communication* 7
2.2 *Wireless Sensor Network* 7
2.3 *Sleep Scheduling* 8
2.4 *Genetic Algorithm* 9
2.5 Darolt WSN Simulator 9
2.6 PyCharm IDE 10
2.7 LEACH 10
2.8 Python 11
2.9 SWIG 11
2.10 C++ 12
BAB III ANALISIS DAN PERANCANGAN **13**

3.1 Deskripsi Umum Sistem 13
3.2 Daftar Istilah 15
3.3 Area 16
3.4 Penandaan *Pixel* 16
3.5 Pembentukan Region 17
3.5.1 Exclusive Region 18
3.5.2 *Overlapping Region* 18
3.6 Penghitungan Luas Region 19
3.7 *Genetic Algorithm* dalam *Sleep Scheduling* 19
3.7.1 Pembuatan Individu 20
3.7.2 Pemilihan Individu 21
3.7.3 *Crossover* 21
3.7.4 *Mutation* 21
3.7.5 *Fitness* 22
BAB IV IMPLEMENTASI 25
4.1 Lingkungan Implementasi 25
4.1.1 Lingkungan Implementasi Perangkat Keras 25
4.1.2 Lingkungan Implementasi Perangkat Lunak 25
4.2 Implementasi 26
4.2.1 Pembuatan Kelas Optimize 26
4.2.2 Pembuatan Kelas Regions 28
4.2.3 Pembuatan Kelas GeneticAlgo 30
4.2.4 Modifikasi Kelas Network 35
BAB V UJI COBA DAN EVALUASI 37
5.1 Lingkungan Pengujian 37
5.1.1 Perangkat Keras 37
5.1.2 Perangkat Lunak 37
5.2 Simulator 38
5.3 Skenario Uji Coba 38
5.3.1 Skenario Uji Coba 1 39
5.3.2 Skenario Uji Coba 2 39
5.3.3 Skenario Uji Coba 3 40
5.4 Hasil Skenario Uji Coba 41
5.4.1 Hasil Skenario Uji Coba 1 41
5.4.2 Hasil Skenario Uji Coba 2 54

5.4.3 Hasil Skenario Uji Coba 3 56
BAB VI KESIMPULAN DAN SARAN 59
6.1 Kesimpulan 59
6.2 Saran 60
DAFTAR PUSTAKA 61
LAMPIRAN 63
BIODATA PENULIS 79

(Halaman ini sengaja dikosongkan)

DAFTAR GAMBAR

Gambar 2.1: Ilustrasi *Wireless Sensor Network* 8
Gambar 2.2: Antarmuka PyCharm IDE..... 10
Gambar 3.1: Skema Umum Sistem 14
Gambar 3.2: Persebaran *Node* dalam Area Berbentuk *Grid*..... 16
Gambar 3.3: *Exclusive Region*..... 18
Gambar 3.4: *Overlapping Region*..... 18
Gambar 3.5: *Flowchart Genetic Algorithm* 20
Gambar 5.1: Jumlah *Node* Aktif Tiap Ronde 41
Gambar 5.2: Penggunaan Energi Tiap Ronde 43
Gambar 5.3: Rata-rata Penggunaan Energi 44
Gambar 5.4: Sisa Energi pada *Node*..... 45
Gambar 5.5: Rata-rata *Coverage* 46
Gambar 5.6: Rata-rata *Overlapping* 47
Gambar 5.7: Hasil Plot *Coverage* Sebelum *Sleep Scheduling*... 48
Gambar 5.8: Hasil Plot *Coverage* Setelah *Sleep Scheduling* pada Ronde 1 49
Gambar 5.9: Hasil Plot *Coverage* Setelah *Sleep Scheduling* pada Ronde 2 50
Gambar 5.10: Hasil Plot *Coverage* setelah *Sleep Scheduling* pada Ronde 3 51
Gambar 5.11: Hasil Plot *Coverage* setelah *Sleep Scheduling* pada Ronde 4 52
Gambar 5.12: Hasil Plot *Coverage* setelah *Sleep Scheduling* pada Ronde 5 53

(Halaman ini sengaja dikosongkan)

DAFTAR TABEL

Tabel 3.1 Daftar Istilah 15
Tabel 5.1 Lingkungan Simulator 32
Tabel 5.2 Konfigurasi WSN pada Skenario 1 39
Tabel 5.3 Konfigurasi WSN pada Skenario 2 40
Tabel 5.4 Konfigurasi WSN pada Skenario 3 40
Tabel 5.5 Perbandingan *Node* Tersisa dan Urutan Ronde Ketika *Node* Mati Pertama Kali 42
Tabel 5.6 Hasil Pengujian Skenario 2 54
Tabel 5.7 Hasil Pengujian Skenario 3 56

(Halaman ini sengaja dikosongkan)

KODE SUMBER

Kode Sumber 3.1 Tahap Cek *Pixel* 16
Kode Sumber 3.2 Tahap Menandai *Pixel* 17
Kode Sumber 3.3 Perhitungan Jarak *Pixel* ke *Node* 17
Kode Sumber 3.4 Pembentukan *Region* 17
Kode Sumber 3.5 Penghitungan Luas *Region* 17
Kode Sumber 4.1 Kelas Optimize 26
Kode Sumber 4.2 Kelas Regions 28
Kode Sumber 4.3 Kelas GeneticAlgo 30
Kode Sumber 4.3 Fungsi get_active_node() 35

(Halaman ini sengaja dikosongkan)

BAB I
PENDAHULUAN

1.1 Latar Belakang

Perkembangan teknologi komunikasi saat ini teleh berkembang dengan sangat pesat, salah satunya dibidang *Wireless Sensor Network* (WSN). WSN adalah salah satu jaringan nirkabel yang terdiri dari banyak *node* sensor yang bekerja secara kooperatif memantau lingkungan yang besar. *Wireless sensor network* terbentuk dari sekumpulan *node* yang diletakkan tersebar pada suatu area dan masing masing *node* tidak hanya terhubung satu sama lain namun juga terhubung pada *base station*. Kemampuan *sensor node* pada suatu wireless sensor network bisa berbeda-beda, seperti perbedaan dari kemampuan *node* dalam menerima rangsangan, seperti *sensor node* sederhana yang hanya bisa menerima satu rangsangan saja hingga *sensor node* kompleks yang bisa menerima banyak rangsangan. Perbedaan lain adalah dari kemampuan *sensor node* dalam komunikasi melalui bermacam-macam cara seperti melalui *ultrasound, infrared,* atau frekuensi radio.

Sensor node yang berjalan pada WSN beroperasi menggunakan baterai, yang artinya energi pada *sensor node* sangat terbatas. Umumnya, *sensor node* berjalan dengan melakukan tiga kegiatan yaitu mendeteksi, berkomunikasi, dan komputasi. Ketiga kegiatan tersebut membutuhkan energi untuk berjalan. Jika semua *node* menyala tanpa melakukan transmisi apapun maka terjadi pemborosan energi. Jadi salah satu cara untuk menghemat energi adalah mengatur tiap-tiap *sensor node* agar tidak selalu aktif secara bersamaan. Salah satu metode yang bisa digunakan adalah *sleep scheduling*.

Sleep Scheduling adalah salah satu mekanisme yang sering digunakan dalam *wireless sensor network* (WSN) dengan tujuan untuk menghemat energi dengan mengubah *state* suatu *node* yang tidak dibutuhkan dari *idle* menjadi *sleep*. Proses *sleep scheduling*

pada umumnya dilakukan dengan membagi rata *node* dalam jadwal-jadwal yang berbeda. Namun cara ini memiliki masalah yaitu berkurangnya jangkauan jaringan secara keseluruhan yang kemudian dapat membuat data yang dikirim menjadi tertunda. Maka dalam tugas akhir ini akan mengimplementasikan *genetic algorithm* pada *sleep scheduling*, diharapkan agar konfigurasi *node* yang diubah menjadi *sleep* dan *alive* tidak hanya menghemat lebih banyak energi dan memperpanjang waktu aktif *node*, namun juga menjaga luas jangkauan deteksi.

1.2 Rumusan Masalah

Rumusan masalah yang diangkat dalam tugas akhir ini dapat dipaparkan sebagai berikut:

1. Bagaimana cara kerja *sleep scheduling* pada *wireless sensor network* (WSN)?
2. Bagaimana penerapan *genetic algorithm* dalam optimasi *sleep scheduling*?

1.3 Batasan Permasalahan

Berdasarkan masalah yang diuraikan oleh penulis, maka batasan masalah pada tugas akhir ini adalah:

1. Simulator yang digunakan adalah simulator WSN berbasis Python.
2. *Node* sensor yang berjalan maksimal berjumlah 300 *node*.
3. Maksimal ronde pengujian WSN sebanyak 2000 ronde.
4. *Genetic Algorithm* berjalan dengan melibatkan 100 gen dan diiterasikan sebanyak 100 kali.

1.4 Tujuan

Tujuan dari pembuatan tugas akhir ini adalah mengoptimalkan proses *sleep scheduling* menggunakan *genetic algorithm* pada *wireless sensor network*.

1.5 Manfaat

Manfaat yang diperoleh dari pengerjaan tugas akhir ini adalah:

1. Mengurangi penggunaan energi pada setiap ronde.
2. Memperpanjang masa hidup setiap *node* pada *wireless sensor network*.

1.6 Metodologi

Pembuatan tugas akhir ini dilakukan dengan menggunakan metodologi sebagai berikut:

1.6.1 Penyusunan Proposal Tugas Akhir

Proposal tugas akhir ini berisi gambaran tentang tugas akhir yang akan dibuat. Pendahuluan proposal tugas akhir meliputi hal yang menjadi latar belakang diajukannya usulan tugas akhir, rumusan masalah yang diangkat, batasan masalah yang menjadi konstrain dari tugas akhir, tujuan pembuatan tugas akhir, dan manfaat dari hasil tugas akhir. Di dalam proposal tugas akhir juga dijabarkan mengenai tinjauan pustaka yang menjadi referensi pendukung dalam pembuatan tugas akhir ini. Terdapat pula sub bab jadwal kegiatan yang menjelaskan jadwal pengerjaan tugas akhir.

1.6.2 Studi Literatur

Studi literatur yang dilakukan dalam pengerjaan Tugas Akhir ini adalah mengenai Implementasi *Genetic Algorithm* untuk Optimasi *Sleep Scheduling* pada *Wireless Sensor Network*. Sehingga, studi literatur ini dapat diterapkan pada penentuan *sleep scheduling* yang tepat dalam jalannya *wireless sensor network*.

1.6.3 Implementasi Sistem

Implementasi merupakan tahap untuk mengimplementasikan metode-metode yang sudah diajukan pada proposal tugas akhir. Untuk membangun algoritma yang telah dirancang sebelumnya, implementasi dilakukan dengan menggunakan simulator WSN yang dibuat menggunakan bahasa pemrograman Python.

1.6.4 Pengujian dan Evaluasi

Pengujian dilakukan untuk menguji apakah *sleep scheduling* yang berjalan bisa memberi pengaruh pada penggunaan energi tiap *sensor node.*

1.6.5 Penyusunan Buku

Pada tahap ini dilakukan penyusunan buku sebagai dokumentasi dari pelaksanaan tugas akhir yang mencakup seluruh konsep, teori, implementasi, serta hasil yang telah dikerjakan.

1.7 Sistematika Penulisan Laporan

Sistematika penulisan laporan tugas akhir adalah sebagai berikut:

1. Bab I. Pendahuluan
   Bab ini berisi penjelasan mengenai latar belakang, rumusan masalah, batasan permasalahan, tujuan, manfaat, metodologi, dan sistematika penulisan dari pembuatan tugas akhir.
2. Bab II. Tinjauan Pustaka
   Bab ini berisi kajian teori atau penjelasan dari metode, algoritma, *library*, dan *tools* yang digunakan dalam penyusunan tugas akhir ini. Kajian teori yang dimaksud berisi tentang penjelasan singkat mengenai *Wireless Sensor Network, Sleep Scheduling*, dan *Genetic Algorithm.*
3. Bab III. Perancangan Perangkat Lunak dan Perangkat Keras

Bab ini berisi pembahasan mengenai desain dari jaringan sensor nirkabel yang akan dibuat, meliputi arsitektur dan proses perangkat lunak.

4. Bab IV. Implementasi
   Bab ini menjelaskan implementasi yang berbentuk *pseudocode* dari program yang dibuat.
5. Bab V. Pengujian dan Evaluasi
   Bab ini berisi hasil uji coba dan evaluasi dari *sleep scheduling* yang dibuat dengan melihat keluaran yang dihasilkan. Uji coba dilakukan dengan menjalankan simulasi kerja WSN menggunakan *sleep scehduling* dan membandingkan dengan hasil tanpa proses *sleep scheduling*. Selain itu juga dilakukan pengujian dengan mengubah variasi jumlah *node* dan luas area.
6. Bab VI. Kesimpulan dan Saran
   Bab ini merupakan bab yang menyampaikan kesimpulan dari hasil uji coba yang dilakukan, masalah-masalah yang dialami pada proses pengerjaan tugas akhir, dan saran untuk pengembangan tugas akhir ke depannya.
7. Daftar Pustaka
   Bab ini berisi daftar pustaka yang dijadikan literatur dalam tugas akhir.
8. Lampiran
   Dalam lampiran terdapat kode sumber program secara keseluruhan.

(Halaman ini sengaja dikosongkan)

BAB II
TINJAUAN PUSTAKA

Bab ini berisi pembahasan mengenai teori-teori dasar yang digunakan dalam pengerjaan tugas akhir. Penjelasan ini bertujuan untuk memberikan gambaran secara umum terhadap program yang dibuat dan berguna sebagai penunjang dalam pengembangan riset yang berkaitan.

2.1 *Wireless Communication*

Wireless Communication atau komunikasi nirkabel adalah komunikasi yang terjadi melalui jaringan tanpa kabel. Komunikasi nirkabel membutuhkan perangkat yang berperan sebagai *transmitter* atau *receiver* untuk berkomunikasi. Komunikasi nirkabel dapat digunakan pada tempat dimana penggunaan kabel tidak disarankan, seperti pada beberapa bangunan bersejarah atau kereta. [1]

Komunikasi nirkabel tidak membutuhkan uang banyak dalam pembuatan dan pemeliharaannya. Hal ini bisa terjadi karena tidak perlu uang lebih untuk membeli kabel, dan biaya pemeliharaan perangkat komunikasi berbasis radio lebih murah dibandingkan perangkat yang menggunakan kabel. [1]

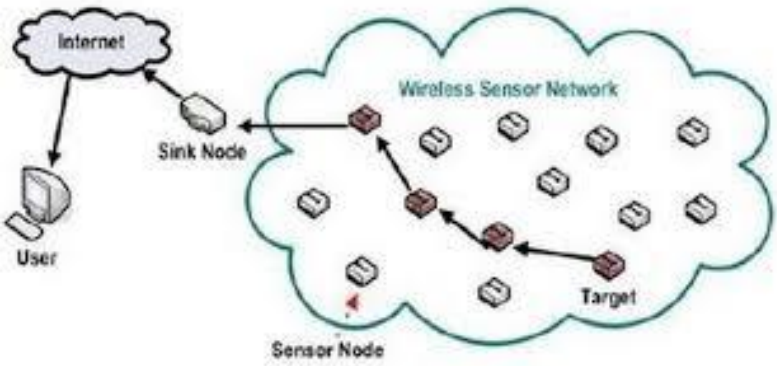
2.2 *Wireless Sensor Network*

Wireless Sensor Network (WSN) adalah jaringan yang terdiri dari beberapa *sensor node* yang tersebar di suatu area dan berkomunikasi mengenai informasi yang didapat dari area yang dimonitor melalui jaringan nirkabel. Data yang dikumpulkan oleh *node* akan diproses dan kemudian dikirimkan menuju *sink* atau *base station.* [2]

WSN terdiri dari *sensor node* yang dilengkapi dengan prosesor, radio nirkabel, sensor, memori, dan baterai. Prosesor digunakan untuk melakukan pemrosesan data. Sensor digunakan

untuk menerima informasi dari area yang dimonitor seperti temperatur, cahaya, kelembaban, dan sebagainya. Memori dibutuhkan untuk menyimpan instruksi yang diproses oleh prosesor dan data yang didapat dari sensor atau data hasil pemrosesan. [2]

WSN memiliki banyak keuntungan seperti biaya yang murah, pengembangan teknologi yang maju, dan bisa diterapkan dalam berbagai bidang. Beberapa bidang yang menggunakan WSN yaitu bidang militer, lingkungan, kesehatan, dan keamanan. [2]



Gambar 2.1 Ilustrasi *Wireless Sensor Network*

2.3 *Sleep Scheduling*

Sleep scheduling adalah teknik yang digunakan untuk menghemat konsumsi baterai pada *wireless sensor network* dengan melakukan penjadwalan *sleep/wake up* pada tiap-tiap *node*. *Sleep scheduling* dilakukan dengan mengubah *state* beberapa *node* yang tidak diperlukan menjadi *sleep* agar tidak menghabiskan energi. [3]

Sleep scheduling diperlukan karena WSN berjalan menggunakan baterai, dimana konsumsi daya menjadi bagian yang sangat diperhatikan agar WSN bisa berjalan lebih lama. Namun dalam penentuan *sleep scheduling* harus memperhatikan konektivitas dan jangkauan deteksi pada WSN agar kualitas

layanan tetap bagus meskipun beberapa *node* dalam keadaan *sleep*. [3]

2.4 *Genetic Algorithm (GA)*

Genetic Algorithm adalah salah satu algoritma pencarian dan optimasi dengan memperagakan proses evolusi makhluk hidup melalui seleksi alam. *Genetic algorithm* mengibaratkan populasi yang terdiri dari beberapa individu sebagai kumpulan jawaban dari sebuah masalah. [4]

Secara umum *genetic algorithm* terdiri dari dua proses, yaitu proses pemilihan individual untuk generasi selanjutnya, dan penggantian individual melalui proses *crossover* dan *mutation*. *Crossover* adalah proses pembentukan individu baru dengan melakukan persilangan gen antara dua individu. *Mutation* adalah proses pembentukan individu dengan melakukan mutasi, yaitu mengubah nilai secara acak dari gen suatu individu. *Genetic algorithm* berjalan secara iteratif menghasilkan individu baru yang menjadi pengganti dari beberapa individu sebelumnya yang tidak unggul. [4]

Nilai dari suatu individu diukur melalui nilai *fitness function*. *Fitness function* adalah sebuah kriteria adaptasi yang dibutuhkan oleh individu untuk menyelesaikan sebuah masalah. Individu yang bagus akan dipertahankan sementara individu yang tidak bagus akan diganti dengan individu baru. [4]

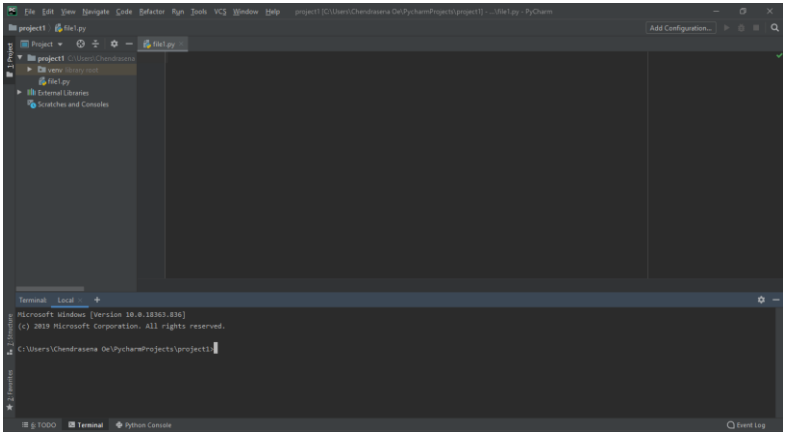
2.5 Darolt WSN Simulator

Darolt WSN Simulator adalah simulator yang mensimulasikan cara kerja *wireless sensor network* seperti komunikasi antar *node* dan komunikasi antara *node* dan *base station*. Pada simulator ini terdapat pengaturan teknik *routing* dan *sleep scheduling*. Teknik *routing* yang ada pada simulator yaitu DC (*Direct Communication*), MTE (*Multi-Threshold Energy*), FCM (*Fuzzy C-Means*), dan LEACH (*Low Energy Adaptive Clustering*

Hierarchy). Untuk *sleep scheduling* pada simulator ini terdapat optimasi menggunakan PSO (*Particle Swarm Optimization*). Simulator ini dibangun menggunakan bahasa pemrograman Python (pada simulasi WSN) dan C++ (pada modul *sleep scheduling*). Modul C++ dijalankan melalui perantara SWIG agar bisa terhubung dan berjalan pada program dengan bahasa pemrograman *Python*. Darolt WSN Simulator dapat diakses melalui *https://github.com/darolt/wsn*.

2.6 PyCharm IDE

PyCharm IDE adalah perangkat lunak yang digunakan sebagai tempat untuk menulis logika, modul, dan perintah dari sebuah program. PyCharm IDE menggunakan bahasa pemrograman Python. PyCharm IDE dikembangkan oleh JetBrains. [5]



Gambar 2.2 Antarmuka PyCharm IDE

2.7 LEACH

LEACH adalah salah satu protokol *routing* dimana semua *node* dikelompokkan menjadi beberapa *cluster*, dan secara

bergiliran memilih *node* menjadi *cluster head* dan sisanya menjadi *cluster member*. *Node* yang menjadi *cluster head* dapat menggunakan energi lebih besar dibandingkan anggota *cluster*. Dalam transmisi data ke *base station*, *node* yang menjadi *cluster member* mengirimkan datanya menuju *cluster head* terdekat, kemudian *cluster head* akan mengirimkan semua data yang diterima dari *cluster member* menuju *base station*. [6]

Penentuan *cluster head* dilakukan dengan menghasilkan nilai acak antara 0 hingga 1 pada tiap node dan membandingkannya dengan nilai *threshold cluster head*. Nilai *threshold cluster head* didapatkan menggunakan rumus:

$$T(n) = \begin{cases} \dfrac{P}{1 - P(r \bmod \frac{1}{P})} & if\ n \in G \\ 0 & if\ n \notin G \end{cases}$$

Dimana P adalah presentase *cluster head*, r adalah ronde saat *threshold* dihitung, n adalah *node*, dan G adalah kumpulan *node* yang tidak menjadi *cluster head* pada ronde sebelumnya. Jika hasil nilai acak *node* lebih kecil dari nilai *threshold*, maka *node* tersebut menjadi *cluster head*. [7]

2.8 Python

Python adalah bahasa pemrograman yang termasuk dalam pemrograman berorientasi objek. Python dapat berjalan pada system operasi Mac OS, Windows, Linux, dan Unix. Hingga saat ini versi terbaru dari bahasa pemrograman Python adalah versi 3.8.3. [8]

2.9 SWIG

SWIG adalah perangkat lunak yang digunakan sebagai alat untuk menghubungkan program yang ditulis dalam bahasa C dan C++ dengan beberapa bahasa pemrograman yang lain, seperti

Python, PHP, Javascript, Perl, dan Ruby. SWIG umumnya digunakan untuk menyusun program, antarmuka, dan sebagai alat yang membantuk dalam proses *testing* perangkat lunak berbahasa C atau C++. [11]

2.10 C++

C++ adalah bahasa pemrograman yang dibuat oleh Bjarne Stroustroup pada tahun 1983 sebagai pengembangan dari bahasa pemrograman C. C++ telah menjadi contoh untuk pengembangan bahasa pemrograman lainnya. [12]

BAB III
ANALISIS DAN PERANCANGAN

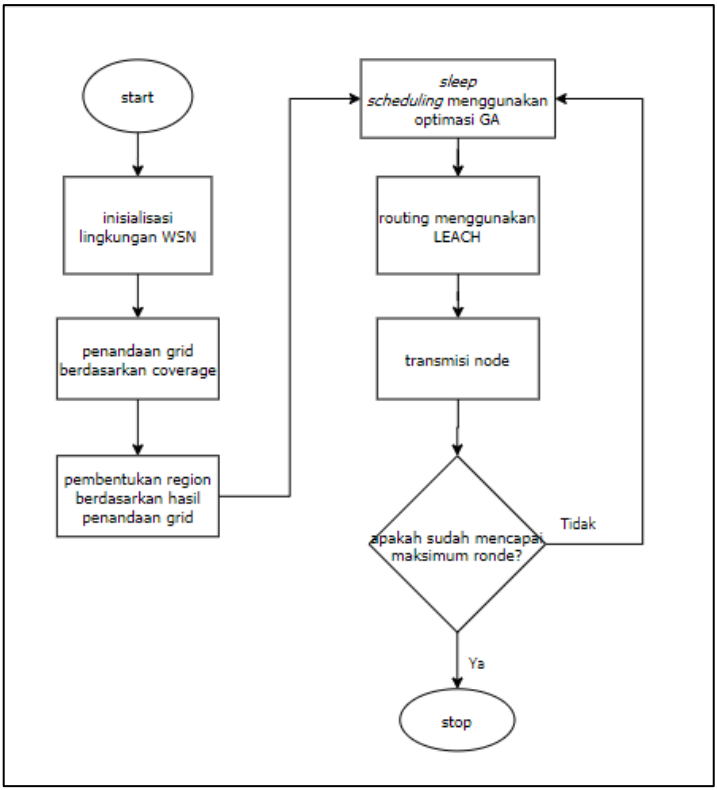
Analisis dan perancangan merupakan bagian penting dalam pembuatan perangkat lunak dan perangkat keras yang berupa perencanaan secara teknis dari sistem jaringan yang dibuat. Pada Bab ini akan dibahas mengenai perancangan dan implementasi *Genetic Algorithm* dalam optimasi *Sleep Scheduling*, dimana hasil optimasi akan menjadi konfigurasi node mana saja yang diubah menjadi sleep dan tidak dalam *Wireless Sensor Network*.

3.1 Deskripsi Umum Sistem

Pada tugas akhir ini akan dibuat sebuah implementasi *genetic algorithm* dalam proses *sleep scheduling* pada *wireless sensor network. Wireless sensor network* (WSN) yang digunakan berupa simulasi perangkat lunak yang dibuat menggunakan bahasa pemrograman Python. WSN berjalan sebanyak 2000 ronde.

Pada tahap awal, dibuatlah 300 *node* dan satu *base station* yang kemudian diletakkan tersebar pada area yang berbentuk *grid*. Khusus untuk letak *base station* akan ditaruh pada titik tengah area. Setelah semua *node* diletakkan maka dilakukan proses pemetaan *coverage* menjadi beberapa *region*. Pembentukan *region* diperlukan dalam perhitungan *fitness* pada *genetic algorithm*.

Di setiap *round*, *sleep scheduling* dijalankan sebelum proses *routing* LEACH. *Sleep scheduling* dihasilkan berdasar pada hasil optimasi *genetic algorithm* dengan memperhatikan tiga faktor, yaitu penggunaan energi, *coverage*, dan *overlapping*. Diharapkan hasil *sleep scheduling* bisa mengurangi penggunaan energi namun tidak mempengaruhi *coverage* secara keseluruhan.



Gambar 3.1 Skema Umum Sistem

Terdapat beberapa asumsi yang berlaku pada berjalannya WSN yaitu:

1. Pada setiap ronde, semua *sensor node* melakukan proses sensing dan selalu mengirimkan data.
2. Energi yang digunakan untuk transmisi pada tiap *sensor node* dapat beragam, tergantung pada jarak suatu *node* dengan *cluster head*. *Node* yang menjadi *cluster head* dapat menggunakan energi lebih besar dibandingkan dengn *sensor node* yang tidak menjadi *cluster head*.

3. Setiap satu ronde, *sensor node* berada pada satu cluster yang sama, dan dapat berganti pada ronde selanjutnya.
4. Tidak ada biaya energi tambahan yang dikeluarkan ketika dilakukan perpindahan *state* dari *sleep* menjadi *active* pada tiap *node*.

3.2 Daftar Istilah

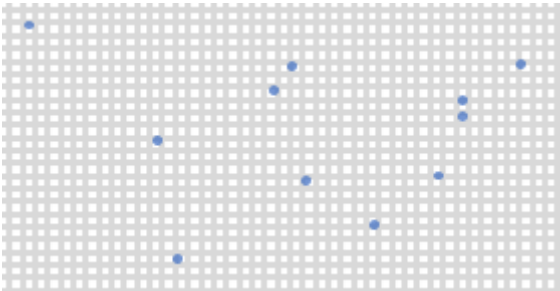
Daftar istilah yang sering digunakan pada buku tugas akhir ini dapat dilihat pada Tabel 3.1.

Tabel 3.1 Daftar Istilah

No.	Istilah	Penjelasan
1	*Node*	Sebuah unit yang menjadi titik koneksi, titik redistribusi, dan titik akhir komunikasi
2	*Node id*	Identitas dari sebuah *node.*
2	*Grid*	Hasil pemetaan suatu area menjadi beberapa *pixel* berbentuk persegi.
3	*Region*	Wilayah yang diawasi oleh *node* dan terdiri dari beberapa *pixel.*
4	*Coverage*	Jangkauan deteksi dari sebuah *node*.
5	*Pixel*	Satuan yang melambangkan satu persegi dalam *grid.*
6	*Routing*	Proses penentuan jalur transmisi data.
7	*Cluster Head*	*Sensor node* yang menjadi kepala pada suatu cluster.
8	*Global fitness*	Individu dengan *fitness* terbaik dari semua individu yang ada.

3.3 Area

Semua *node* akan disebar secara acak dalam area berbentuk *grid*. *Base station* diletakkan pada tengah area. Setiap koordinat (x,y) mewakili 1 pixel.



Gambar 3.2 Persebaran *Node* Dalam Area Berbentuk *Grid*

3.4 Penandaan *Pixel*

Setelah *node* selesai diletakkan pada area, maka selanjutnya dilakukan penandaan *pixel* sebagai tahapan awal pembentukan *region*. Tahap ini dilakukan penandaan tiap *pixel* yang masuk dalam *coverage* tiap-tiap *node*, ditandai dengan *node id*. *Pixel* dapat ditandai dengan banyak *node id* jika ditemukan suatu *pixel* berada pada *coverage* lebih dari satu *node*.

```
for pixel_x in pixels[x]:
  for pixel_y in pixels[y]:
    distance = calculate_distance_point(pixel_x, pixel_y,
    node.pos_x, node.pos_y)
    if distance < coverage_radius then
      paint_pixel(pixel_x, pixel_y, node.id)
```

Kode Sumber 3.1 Tahap Cek *Pixel*

```
Function paint_pixel(pixel_x, pixel_y, node.id):
  if pixels[pixel_x][pixel_y] not null then
    add node.id to pixels[pixel_x][pixel_y]
  else
    pixels[x][y] = [node.id]
```

Kode Sumber 3.2 Tahap Menandai *Pixel*

```
Function calculate_distance_point(pixel_x, pixel_y, node.pos_x, node.pos_y):
  return sqrt((pixel_x - node.pos_x)^2 + (pixel_y - node.pos_y)^2)
```

Kode Sumber 3.3 Perhitungan Jarak *Pixel* Ke *Node*

3.5 Pembentukan Region

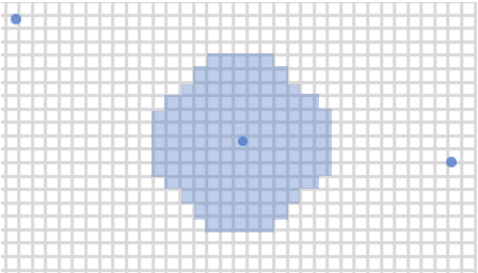
Setelah semua *pixel* yang masuk dalam *coverage* setiap *node* telah ditandai, maka dilakukan tahap pembentukan *region*. Tahap ini dilakukan dengan menggabungkan beberapa *pixel* dengan tanda *node id* yang sama menjadi satu *region*. Terdapat dua jenis *region*, yaitu *exclusive region* dan *overlapping region*.

```
function grid2region(pixels):
  region = []
  for pixel_x in pixels[x]:
    for pixel_y in pixels[y]:
      owners = pixels[pixel_x][pixel_y]
      region = check_region(owner)
      if region not null then
        region.area = region.area + area_single_pixel
      else
        new_region = Region(area_single_pixel, owners)
      add new_region to region
```

Kode Sumber 3.4 Pembentukan Region

3.5.1 Exclusive Region

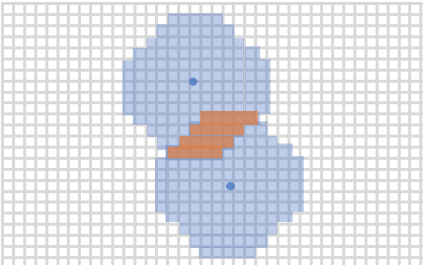
Exclusive region adalah *region* yang melambangkan area yang masuk dalam *coverage* satu *node* yang tidak tumpeng tindih. *Exclusive region* terbentuk dari *pixel* yang ditandai dengan hanya satu *node id*.



Gambar 3.3 *Exclusive Region*

3.5.2 *Overlapping Region*

Overlapping region adalah *region* yang melambangkan area yang masuk dalam *coverage* beberapa *node* atau dapat dikatakan tumpang tindih. *Overlapping region* terbentuk dari *pixel* yang ditandai dengan *node id* lebih dari satu. Pada gambar 3.4 *overlapping region* ditandai dengan warna coklat.



Gambar 3.4 *Overlapping Region*

3.6 Penghitungan Luas Region

Setelah *region* selesai dibuat maka akan dihitung luas semua *region* dan dikategorikan menjadi tiga, yaitu luas *exclusive region*, luas *overlapping region*, dan luas *coverage*. *Luas exclusive region* dihitung dari semua *exclusive region*, *luas overlapping region* dihitung dari semua *overlapping region*, dan *luas coverage* dihitung dari semua *region* baik *exclusive* maupun *overlapping*. Satuan yang digunakan adalah satuan *grid pixel*. Penghitungan luas *region* perlu dilakukan untuk membantu proses penghitungan *fitness* pada *genetic algorithm*.

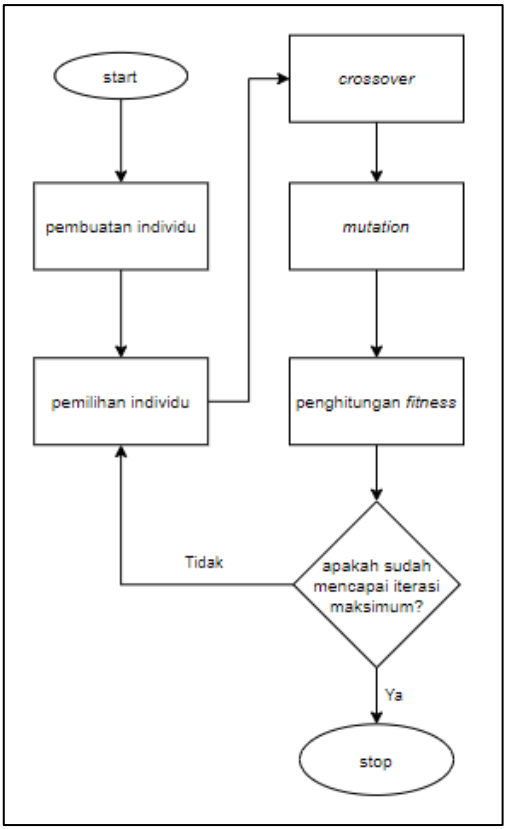
```
total_coverage_exclusive = 0.0
total_coverage_overlapping = 0.0
for x in self.region_exclusive:
  if energies[x] != 0.0:
    total_coverage_exclusive +=
self.region_exclusive[x]

for x in self.region_overlapping:
  for y in x[0]:
    if energies[y] != 0.0:
      total_coverage_overlapping += x[1]
      break

self.total_region_overlapping =
total_coverage_overlapping
self.total_coverage = total_coverage_exclusive +
total_coverage_overlapping
```

Kode Sumber 3.5 Penghitungan Luas *Region*

3.7 *Genetic Algorithm* dalam *Sleep Scheduling*

Setelah *region* selesai dibuat maka langkah selanjutnya adalah proses *sleep scheduling*. *Sleep scheduling* membutuhkan informasi energi yang tersisa pada *node* yang aktif dan semua *region* yang telah dibuat. Hasil penghitungan yang telah dilakukan

sebelumnya diperlukan sebagai parameter penghitungan nilai *fitness* pada *genetic algorithm*. Terdapat beberapa proses yang akan dilakukan, yaitu pembuatan individu, pemilihan individu, *crossover*, *mutation*, dan penghitungan *fitness*.



Gambar 3.5 *Flowchart Genetic Algorithm*

3.7.1 Pembuatan Individu

Langkah awal dalam proses optimasi *genetic algorithm* adalah pembuatan individu. Individu yang dibuat memuat

informasi mengenai konfigurasi *node* dan nilai *fitness* lokal. Konfigurasi yang dibuat berisi nilai 0 dan 1, 0 berarti *active*, 1 berarti *sleep*. Dalam inisiasi awal, konfigurasi *node* dihasilkan secara acak.

3.7.2 Pemilihan Individu

Individu yang telah dibuat akan melewati proses pemilihan, dimana semua individu akan diurutkan berdasarkan nilai *fitness* tertinggi hingga rendah. Dua per lima bagian paling depan dari semua individu akan dipertahankan, sementara sisanya akan diubah dalam proses *crossover* dan *mutation*. Hal ini dilakukan agar dalam proses *crossover* individu yang disilangkan adalah individu dengan nilai *fitness* yang termasuk baik dari tiap generasi.

3.7.3 *Crossover*

Crossover adalah proses persilangan antara dua individu yang menghasilkan individu baru. Dua individu yang akan disilangkan dipilih secara acak dari kumpulan individu yang dipertahankan pada iterasi saat itu. Pemilihan individu yang akan diganti melalui proses *crossover* dilakukan secara acak dengan menghasilkan nilai acak antara 0.0 dan 1.0, yang kemudian dibandingkan dengan *crossover rate*. *Crossover rate* didapatkan melalui rumus:

$$Crossover\ rate = 1.0 - \frac{CurrIter}{MaxIter}$$

Dimana *CurrIter* adalah urutan iterasi saat itu dan *MaxIter* adalah batas maksimum iterasi dalam *genetic algorithm.*

3.7.4 *Mutation*

Mutation adalah proses penggantian individu dengan mengganti nilai gen pada individu secara acak. Banyak nilai pada

individu yang akan diganti ditentukan dari *mutation rate*. *Mutation rate* didapatkan melalui rumus:

$$Mutation\ rate = (\omega_{max} - \omega_{min}) \times \frac{CurrIter}{MaxIter}$$

Dimana $\omega_{max} = 0.9$, $\omega_{min} = 0.4$, *CurrIter* adalah urutan iterasi saat itu, dan *MaxIter* adalah batas maksimum iterasi pada *genetic algorithm*.

Terdapat mekanisme ketika *global fitness* tidak berubah hingga 3 iterasi maka nilai probabilitas mutation menjadi 1.0, dalam arti *mutation* pasti dilakukan pada semua individu yang diganti. Hal ini dilakukan agar individu yang dibuat lebih beragam dan diharapkan lebih cepat dalam menemukan konfigurasi yang lebih baik.

3.7.5 *Fitness*

Nilai *fitness* pada *genetic algorithm* melambangkan tingkat kesesuaian suatu individu terhadap kriteria yang diinginkan. Kriteria yang diinginkan pada *sleep scheduling* adalah hasil konfigurasi dapat mengurangi penggunaan energi, luas *coverage* tetap sama, dan luas area *overlapping* dapat dikurangi. Pada perhitungan *fitness*, ketiga kriteria memiliki bobot yang sama. Nilai *fitness* didapatkan melalui rumus:

$$fit = \left(1.0 - \frac{\sum_{i=1}^{K} E_{partial}}{\sum_{i=1}^{K} E_{total}}\right) + \left(\frac{C_{partial}}{C_{total}}\right) + \left(1.0 - \frac{O_{partial}}{O_{total}}\right)$$

Dimana K adalah jumlah *node*, $E_{partial}$ adalah total energi dari semua *node* yang tetap aktif saat penerapan *sleep scheduling*, E_{total} adalah total energi dari semua *node* yang aktif, $C_{partial}$ adalah luas total *coverage* setelah penerapan *sleep scheduling*, C_{total} adalah luas total *coverage* sebelum penerapan *sleep scheduling*, $O_{partial}$ adalah luas area *overlapping* setelah penerapan *sleep scheduling*, dan O_{total} adalah luas area *overlapping* sebelum penerapan *sleep scheduling*.

Pada Darolt WSN Simulator sudah dibuat optimasi *sleep scheduling* menggunakan *genetic algorithm*, namun optimasi tersebut memiliki perbedaan, salah satunya pada bagian *fitness*, dimana *overlapping area* tidak menjadi pertimbangan dalam penentuan *fitness*. Jika hal itu dilakukan maka konfigurasi yang dibuat tidak efisien karena banyak *sensor node* yang saling berdekatan, yang berarti *overlapping area* sangat besar dan jangkauan deteksi keseluruhan menjadi lebih kecil.

(Halaman ini sengaja dikosongkan)

BAB IV
IMPLEMENTASI

Bab ini membahas mengenai implementasi sistem yang dilakukan berdasarkan rancangan yang telah dijabarkan pada bab sebelumnya. Implementasi berupa *pseudocode* untuk membangun program.

4.1 Lingkungan Implementasi

Lingkungan Implementasi merupakan lingkungan dimana sistem akan dibangun. Lingkungan implementasi dibagi menjadi Lingkungan Implementasi Perangkat keras dan Lingkungan Implementasi Perangkat Lunak.

4.1.1 Lingkungan Implementasi Perangkat Keras

Pada bagian ini dijelaskan perangkat keras yang digunakan untuk membangun sistem. Spesifikasi komputer yang digunakan dalam perancangan sistem antara lain:

1. Model: Lenovo Z40-75
2. *Processor*: AMD A10-7300, @ 1.9 GHz
3. RAM: 8 GB
4. *Hard Disk Storage*: 1 TB
5. Sistem Operasi: Windows 10

4.1.2 Lingkungan Implementasi Perangkat Lunak

Pada bagian ini akan dibahas mengenai perangkat lunak apa saja yang dibutuhkan untuk membangun sitem. Detail perangkat lunak yang digunakan dalam perancangan sistem antara lain:

1. *Software* IDE: PyCharm IDE 2019.2
2. Bahasa Pemrograman: Python versi 3.8

4.2 Implementasi

Pada bagain implementasi menjelaskan mengenai pembangunan perangkat lunak secara detail dan menampilkan kode sumber yang dibuat dalam implementasi *genetic algorithm*. Implementasi dilakukan dengan membuat tiga kelas untuk membantu dalam proses *sleep scheduling*, yaitu kelas Optimize, kelas Regions, dan kelas GeneticAlgo. Pembuatan kelas tersebut menggunakan bahasa pemrograman yang sama dengan modul WSN yang telah ada pada simulator yaitu Python, hal ini dilakukan untuk meningkatkan performa simulator dimana tidak perlu lagi melakukan proses penghubungan modul-modul optimasi berbahasa C++ menuju modul WSN berbahasa Python melaui SWIG.

4.2.1 Pembuatan Kelas Optimize

Kelas Optimize bertugas sebagai inisiator pembentukan objek Regions dan GeneticAlgo dan mencatat rincian hasil *genetic algorithm*. Rincian yang dimaksud adalah konfigurasi *node* terbaik, nilai *fitness* terbaik, luas *coverage* keseluruhan dari konfigurasi terbaik, dan luas *overlapping region* dari konfigurasi terbaik. Optimize membutuhkan data konfigurasi *wireless sensor network*, semua *overlapping region*, semua *exclusive region*, dan rincian energi yang tersisa pada tiap *node*.

```
1   class Optimize:
2     learning[]
3     coverage
4     overlapping
5     term1[]
6     term2[]
7     term3[]
8     bestConfig[]
9     total_energy
10    config[]
11    nodes[]
```

```
  region

  function __init__(exclusive, overlapping, ids, configuration):
    config <- configuration
    nodes <- ids
    region <- new Regions(exclusive, overlapping)

  function Run(energies):
    ClearLearningTraces()
    InitializeSession(energies)
    ga <- new GeneticAlgo(energies, nodes, total_energy, config, region)
    bestConfig <- ga.getBestConfig()
    learning <- ga.getLearningTrace()
    coverage <- ga.getBestCoverage()
    overlapping <- ga.getBestOverlapping()
    term1 <- ga.getBestTerm1()
    term2 <- ga.getBestTerm2()
    term3 <- ga.getBestTerm3()
    delete ga
    return bestConfig

  function InitializeSession(energies):
    energy <- energies
    total_energy <- 0.0
    for x in energy:
      total_energy_temp <- total_energy_temp + x
    total_energy <- total_energy_temp
    region.InitSession(energy)

  function ClearLearningTraces():
    learning.clear()
    coverage <- 0.0
    overlapping <- 0.0
    term1.clear()
    term2.clear()
    term3.clear()
    bestConfig.clear()

  function GetLearningTrace():
    return learning
```

```
51
52    function GetBestCoverage():
53      return coverage
54
55    function GetTerm1Trace():
56      return term1
57
58    function GetTerm2Trace():
59      return term2
60
61    function GetTerm3Trace():
62      return term3
63
64    function GetBestOverlapping():
65      return overlapping
66
```

Kode Sumber 4.1 Kelas Optimize

4.2.2 Pembuatan Kelas Regions

Kelas *Regions* bertugas sebagai kelas yang menangani operasi hitung yang melibatkan *region* yang telah dibuat sebelumnya, seperti perhitungan luas *overlapping region*, luas *coverage*, dan luas *exclusive region* ketika konfigurasi *sleep scheduling* diimplementasikan.

```
1   class Regions:
2     total_region_exclusive
3     total_region_overlapping
4     total_coverage
5     region_exclusive[]
6     region_overlapping[]
7
8     function __init__(exclusive, overlapping):
9       region_exclusive <- exclusive
10      region_overlapping <- overlapping
11      for x in exclusive:
12        total_region_exclusive <-
    total_region_exclusive + exclusive[x]
13
```

```
  function InitSession(energies):
    total_coverage_exclusive <- 0.0
    total_coverage_overlapping <- 0.0
    for x in region_exclusive:
      if energies[x] > 0.0 then
        total_coverage_exclusive <- total_coverage_exclusive + region_exclusive[x]
    total_region_exclusive <- total_coverage_exclusive
    for x in region_overlapping:
      for y in x[0]:
        if energies[y] != 0.0 then
          total_coverage_overlapping <- total_coverage_overlapping + x[1]
        break
      total_region_overlapping <- total_coverage_overlapping
      total_coverage <- total_coverage_exclusive + total_coverage_overlapping

  function getCoverage(individuals, energies):
    partial_overlapping <- 0.0
    partial_exclusive <- 0.0
    partial_coverage <- total_coverage
    for x in region_exclusive:
      if individuals[x] == 0 and energies[x] != 0.0 :
        partial_exclusive <- partial_exclusive + region_exclusive[x]

    for ov in range(0, len(region_overlapping)):
      remaining_owner <- 0
      for owner in region_overlapping[ov][0]:
        if individuals[owner] == 0 and energies[owner] != 0.0 :
          remaining_owner <- remaining_owner + 1
          if remaining_owner > 1:
            break
      if remaining_owner == 1:
        partial_exclusive <- partial_exclusive + region_overlapping[ov][1]
      else if remaining_owner > 1:
```

```
        partial_overlapping <- partial_overlapping +
region_overlapping[ov][1]

    partial_coverage <- partial_overlapping +
partial_exclusive
    coverage_info[]
    coverage_info.append(partial_exclusive)
     coverage_info.
append(total_region_exclusive)
     coverage_info.append(partial_overlapping)
     coverage_info.
append(total_region_overlapping)
    coverage_info.append(partial_coverage)
    coverage_info.append(total_coverage)

    return coverage_info
```

Kode Sumber 4.2 Kelas Regions

4.2.3 Pembuatan Kelas GeneticAlgo

Kelas GeneticAlgo bertugas untuk melakukan proses optimasi *genetic algorithm* yaitu pembuatan individu, *crossover*, *mutation*, dan penghitungan *fitness*. Informasi yang dihasilkan oleh kelas ini adalah konfigurasi terbaik beserta detilnya seperti *coverage* terbaik, luas total *overlapping region* terbaik, dan nilai *fitness* terbaik.

```
Class GeneticAlgo:
  globalFitness
  bestConfig[]
  bestTerm1
  bestTerm2
  bestTerm3
  bestCoverage
  bestOverlapping
  function __init__(energies, ids, total_energy,
config, regions):
    max_individual <- config['NB_INDIVIDUAL']
    max_iter <- config['MAX_ITERATION']
    wmax <- config['WMAX']
```

```
    wmin <- config['WMIN']
    alpha <- config['FITNESS_ALPHA']
    beta <- config['FITNESS_BETA']
    gamma <- config['FITNESS_GAMMA']
    lod <- 0
    energy <- energies
    total_energy <- total_energy
    individual <- createIndividual(ids, config, regions)
    learning <- [globalFitness, bestTerm1, bestTerm2, bestTerm3, bestcoverage, bestoverlapping]
    for x in range(1, max_iter+1):
      individual.sort(x in descending)
      prev_globalfitness <- globalFitness
      crossover_rate <- 1.0 - (x / max_iter)
      if lod >= 3 then
        mutation_rate <- 1.0
        lod <- 0
      else
        mutation_rate <- wmax - ((wmax - wmin) * (x / max_iter))
      nb_unfit <- 0.6*max_individual
      for id in range(int(nb_unfit), len(individual)):
        crossover_prob <- random.uniform(0.0, 1.0)
        if crossover_prob < crossover_rate then
          crossover(id)
        mutate(id, mutation_rate)
      for x in range(0, len(individual)):
        fitness <- Fitness(individual[x][0], regions)
        newgene <- (list(individual[x][0]), fitness[0])
        individual[x] <- newgene
        if fitness[0] > globalFitness then
          globalFitness <- fitness[0]
          bestconfig <- individual[x][0]
          bestTerm1 <- fitness[1]
          bestTerm2 <- fitness[2]
          bestTerm3 <- fitness[3]
          bestcoverage <- fitness[4]
```

```
            bestoverlapping <- fitness[5]
        if prev_globalfitness == globalFitness then
          lod = lod + 1

      learning_final <- [globalFitness, bestTerm1, bestTerm2, bestTerm3, bestcoverage, bestoverlapping]
      learning.extend(learning_final)

    function createIndividual(ids, config, regions):
      individual[]
      nodes <- max_individual
      for x in range(0, nodes):
        config[]
        for y in ids:
          random_val <- random.uniform(0.0, 1.0)
          if energy[y] == 0.0  then
            status <- 1
          else if random_val < 0.5 then
            status <- 1
          else
            status <- 0
          config.append(status)
      fitness_local <- Fitness(config, regions)
      if fitness_local[0] > globalFitness then
        globalFitness <- fitness_local[0]
        bestconfig <- config
        bestTerm1 <- fitness_local[1]
        bestTerm2 <- fitness_local[2]
        bestTerm3 <- fitness_local[3]
        bestcoverage <- fitness_local[4]
        bestoverlapping <- fitness_local[5]
      individual.append((list(config), fitness_local[0]))
      return individual

    function crossover(id):
      father_id <- random.randint(0, id)
      mother_id <- random.randint(0, id)
      father <- individual[father_id][0]
      mother <- individual[mother_id][0]
      top_rate <- random.uniform(0.0, 1.0)
```

```
    new_gene[]
    for x in range(0, len(father)):
        if top_rate >= 0.5 then
            if x < (len(father) // 2) then
                new_gene.append(father[x])
            else
                new_gene.append(mother[x])
        else
            if x < (len(father) // 2) then
                new_gene.append(mother[x])
            else
                new_gene.append(father[x])
    temp <- (list(new_gene), individual[id][1])
    individual[id] <- temp

  function mutate(id, mutation_rate):
    val_changes <- int(float(len(individual[id][0]))*mutation_rate)
    changed_gene <- individual[id][0]
      for count in range(0, val_changes):
        changed_id <- int(random.uniform(0, len(individual[id][0])-1))
          if changed_gene[changed_id] == 0 then
            changed_gene[changed_id] <- 1
          else if changed_gene[changed_id] == 1 then
            changed_gene[changed_id] <- 0
      temp <- (list(changed_gene), individual[id][1])
      individual[id] <- temp

  function Fitness(gene, regions):
    genes <- gene
    partial_energy <- 0.0
    for x in range(0, len(genes)):
      if energy[x] > 0.0 and genes[x] == 0 then
        partial_energy = partial_energy + energy[x]
    coverage_info <- regions.getCoverage(genes, energy)
    term1 <- 0.0
    term2 <- 0.0
    term3 <- 0.0
```

```
    if total_energy != 0.0 then
      term1 <- 1.0 - (partial_energy / total_energy)
    if coverage_info[5] != 0.0 then
      term2 <- abs(coverage_info[4]) / abs(coverage_info[5])
    if coverage_info[3] != 0.0 then
      term3 <- 1.0 - (abs(coverage_info[2]) / abs(coverage_info[3]))
    fitness_val <- (alpha * term1) + (beta * term2) + (gamma * term3)
    fitness_all <- [fitness_val, term1, term2, term3, coverage_info[4], coverage_info[2]]
    return fitness_all

  function getBestConfig():
    return bestconfig

  function getLearningTrace():
    fitness_best <- [learning[0], learning[6]]
    return fitness_best

  function getBestTerm1():
    term1_best <- [learning[1], learning[7]]
    return term1_best

  function getBestTerm2():
    term2_best <- [learning[2], learning[8]]
    return term2_best

  function getBestTerm3():
    term3_best <- [learning[3], learning[9]]
    return term3_best

  function getBestCoverage():
    return learning[10]

  function getBestOverlapping():
    return learning[11]

```

Kode Sumber 4.3 Kelas GeneticAlgo

4.2.4 Modifikasi Kelas Network

Pada Darolt WSN Simulator, penerapan *routing* pada WSN dilakukan sebelum *sleep scheduling* berjalan. *Node* yang masuk dalam proses *routing* adalah semua *node* yang masih memiliki sisa energi. Hal ini dapat mempengaruhi kualitas *routing* jika *sleep scheduling* dijalankan karena jumlah *child node* yang sudah dibagi tiap *cluster* akan berkurang. Modifikasi dilakukan dengan mengubah *node* yang digunakan dalam *routing*, dari semua *node* yang memiliki sisa energi menjadi *node* yang memiliki sisa energi dan dalam keadaan aktif (tidak dalam keadaan *sleep*). Modifikasi dilakukan dengan membuat fungsi get_active_nodes().

```
1  def get_active_nodes(self):
2    is_active = lambda x: x.alive and not
3  x.is_sleeping
4    return [node for node in self[0:-1] if
5  is_active(node)]
```

Kode Sumber 4.4 Fungsi get_active_node()

(Halaman ini sengaja dikosongkan)

BAB V
UJI COBA DAN EVALUASI

Bab ini berisi penjelasan mengenai scenario uji coba yang dilakukan dan evaluasi terhadap performa *sleep scheduling*. Hasil uji coba didapatkan dari implementasi yang dijelaskan pada Bab 4 dengan skenario yang berbeda. Bab ini berisikan pembahasan mengenai lingkungan pengujian data pengujian, dan uji kinerja.

5.1 Lingkungan Pengujian

Lingkungan pengujian sistem merupakan lingkungan dimana sistem akan dijalankan. Pada bagian ini, untuk mempermudah akan dibagi menjadi dua bagian yaitu perangkat keras dan perangkat lunak.

5.1.1 Perangkat Keras

Pada bagian ini akan dibahas mengenai perangkat keras apa saja yang digunakan dalam menjalankan sistem. Spesifikasi komputer yang digunakan dalam menjalankan sistem ini, antara lain :

Model : Lenovo Z40-75
Processor : AMD A10-7300, @ 1.9 GHz
RAM : 8 GB
Hard Disk Storage : 1 TB
Sistem Operasi : Windows 10

5.1.2 Perangkat Lunak

Pada bagian ini akan dibahas mengenai perangkat lunak apa saja yang digunakan dalam perancangan sistem. Detail perangkat lunak yang digunakan diantaranya :

1. *Software* IDE : PyCharm IDE 2019.2
2. Bahasa Pemrograman : Python 3.8

5.2 Simulator

Lingkungan simulator berupa ketentuan – ketentuan yang berupa ukuran – ukuran spesifikasi dari WSN yang disimulasikan. Secara umum lingkungan simulator dijelaskan lebih lanjut pada tabel 5.1 berikut:

Tabel 5.1 Lingkungan Simulator

Keterangan	Detail
Maksimum ronde	2000
Coverage radius (grid pixel)	15
Panjang pesan (*bits*)	4000
Panjang header (*bits*)	150
Energi awal tiap *node* (*Joule*)	2

5.3 Skenario Uji Coba

Sebelum melakukan uji coba, perlu ditentukan skenario yang akan digunakan dalam proses uji coba. Dengan skenario yang dibuat kita akan menguji apakah *sleep scheduling* yang dibuat mampu memberikan dampak atau tidak. Skenario pengujian dibagi menjadi tiga, yaitu skenario dengan *sleep scheduling* dan tanpa *sleep scheduling*, skenario variasi jumlah *node*, dan variasi luas area.

5.3.1 Skenario Uji Coba 1

Pada skenario uji coba 1 ini terdapat 2 metode yang diujikan, yaitu WSN menggunakan *routing* LEACH dengan *sleep scheduling*, dan WSN menggunakan *routing* LEACH tanpa *sleep scheduling*. Terdapat beberapa aspek yang diperhatikan, yaitu aspek *node*, energi, *coverage*, dan *overlapping*. Pengujian dilakukan satu kali untuk masing-masing metode.

Tabel 5.2 Konfigurasi WSN pada Skenario 1

Keterangan		Detail
Area	Panjang area (*grid pixel*)	250
	Lebar area (*grid pixel*)	250
Node	Jumlah *node*	300
Genetic algorithm	Banyak individu	100
	Banyak iterasi	100

5.3.2 Skenario Uji Coba 2

Pada skenario uji coba 2 ini terdapat 3 variasi jumlah *node* yang diujikan, yaitu 50 *node*, 150 *node*, dan 300 *node*. Terdapat beberapa aspek yang dibandingkan, yaitu aspek penggunaan energi, *coverage* sebelum dan sesudah penerapan *sleep scheduling*, dan luas *overlapping area* sebelum dan sesudah penerapan *sleep scheduling*. Pengujian dilakukan satu kali untuk masing-masing metode.

Tabel 5.3 Konfigurasi WSN pada Skenario 2

Keterangan		Detail
Area	Panjang area (*grid pixel*)	250
	Lebar area (*grid pixel*)	250
Node	Jumlah *node*	50/150/300
Genetic algorithm	Banyak individu	100
	Banyak iterasi	100

5.3.3 Skenario Uji Coba 3

Pada skenario uji coba 3 ini terdapat 3 variasi luas area yang diujikan, yaitu 100x100, 150x150, dan 250x250. Terdapat beberapa aspek yang dibandingkan, yaitu aspek penggunaan energi, *coverage* sebelum dan sesudah penerapan *sleep scheduling*, dan luas *overlapping area* sebelum dan sesudah penerapan *sleep scheduling*. Pengujian dilakukan satu kali untuk masing-masing metode.

Tabel 5.4 Konfigurasi WSN pada Skenario 3

Keterangan		Detail
Area	Panjang area (*grid pixel*)	100/150/250
	Lebar area (*grid pixel*)	100/150/250
Node	Jumlah *node*	100

Genetic algorithm	Banyak individu	100
	Banyak iterasi	100

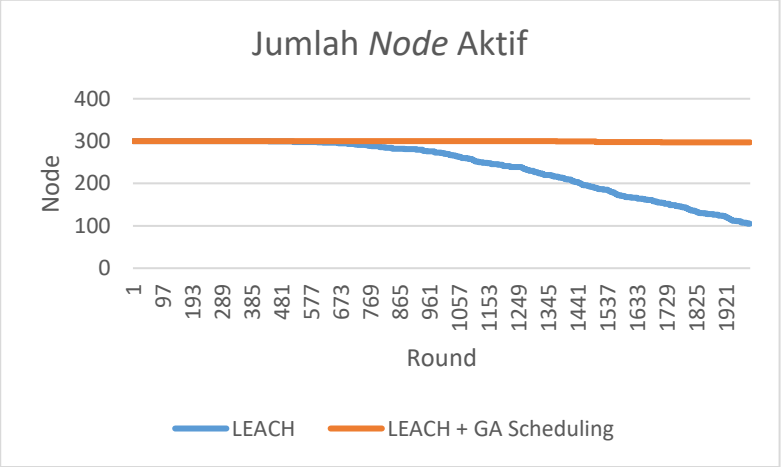
Terdapat perbedaan jumlah *node* pada skenario ini yaitu 100 *node*. Penentuan jumlah *node* tersebut untuk menghindari terbentuknya area *overlapping* yang terlalu besar pada luas area 100x100.

5.4 Hasil Skenario Uji Coba

Berikut adalah hasil uji coba yang telah dilakukan pada tiap skenario:

5.4.1 Hasil Skenario Uji Coba 1

Berikut hasil skenario uji coba antara WSN menggunakan *routing* LEACH dengan *sleep scheduling* dan WSN menggunakan *routing* LEACH tanpa *sleep scheduling*:



Gambar 5.1 Jumlah *Node* Aktif Tiap Ronde

Berdasarkan hasil yang ditunjukkan pada gambar 5.1 terlihat perbedaan yang cukup jauh dimana jumlah *node* yang aktif tiap ronde pada WSN menggunakan *routing* LEACH dan GA *scheduling* jauh lebih banyak dibandingkan WSN menggunakan *routing* LEACH tanpa *sleep scheduling*.

Pada gambar 5.1, kurva WSN menggunakan *routing* LEACH dan GA *scheduling* mencerminkan jumlah *node* yang tersisa pada WSN menggunakan *routing* LEACH dan GA *scheduling* mengalami penurunan yang sangat sedikit karena dalam penggunaaan energi tiap ronde yang lebih kecil dan tidak semua *node* aktif pada tiap ronde. Sedangkan kurva pada WSN menggunakan *routing* LEACH tanpa *sleep scheduling* mencermikan jumlah *node* yang tersisa pada WSN menggunakan *routing* LEACH tanpa *sleep scheduling* mengalami penurunan yang cukup cepat karena pada setiap ronde semua *node* dalam keadaan aktif, dengan begitu energi yang digunakan tiap ronde semakin besar.

Tabel 5.5 Perbandingan Node Tersisa dan Urutan Ronde Ketika Node Mati Pertama Kali

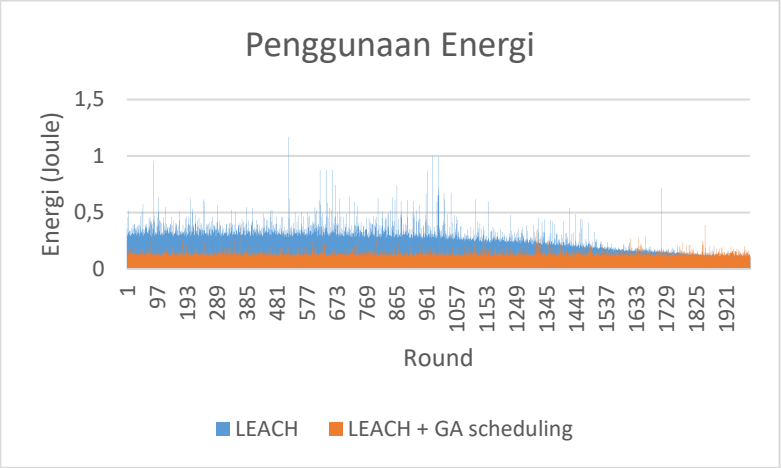
	LEACH	LEACH + GA *Scheduling*
Node tersisa	105	297
Urutan ronde ketika *node* mati pertama kali	440	1371

Dari tabel 5.5 *node* mati pertama kali pada lingkungan WSN dengan routing LEACH tanpa *sleep scheduling* terjadi pada ronde 440, sedangkan pada lingkungan WSN dengan *sleep scheduling* menggunakan *genetic algorithm* terjadi pada ronde 1371. Hal ini disebabkan karena semua *node* pada WSN dengan

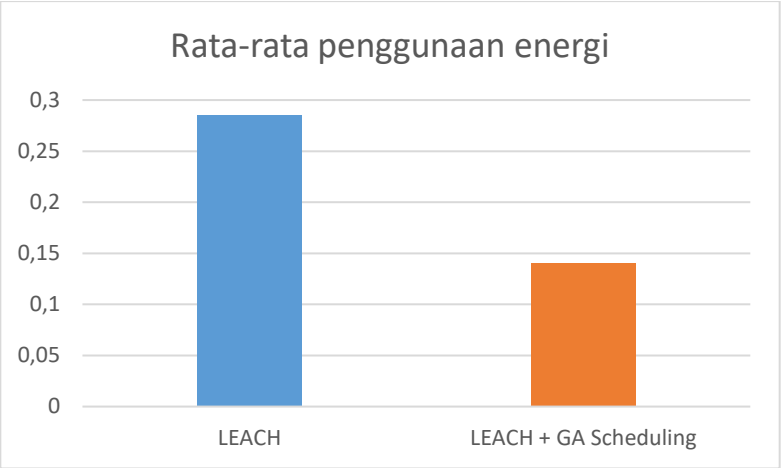
routing LEACH tanpa *sleep scheduling* dalam keadaan aktif melakukan deteks dan transmisi.

Node yang tersisa hingga akhir ronde pada lingkungan WSN tanpa *sleep scheduling* berjumlah 105, lebih sedikit dibandingkan dengan jumlah *node* pada lingkungan WSN dengan *sleep scheduling* menggunakan *genetic algorithm* yang berjumlah 297. Hal ini disebabkan penggunaan energi yang lebih banyak pada WSN dengan *routing* LEACH tanpa *sleep scheduling*.

Dengan demikian dapat menunjukkan pada lingkungan WSN dengan *sleep scheduling* menggunakan *genetic algorithm*, jumlah *node* yang aktif lebih banyak dan masa aktif *node* lebih lama dibandingkan lingkungan WSN dengan routing LEACH tanpa *sleep scheduling*.



Gambar 5.2 Penggunaan Energi Tiap Ronde

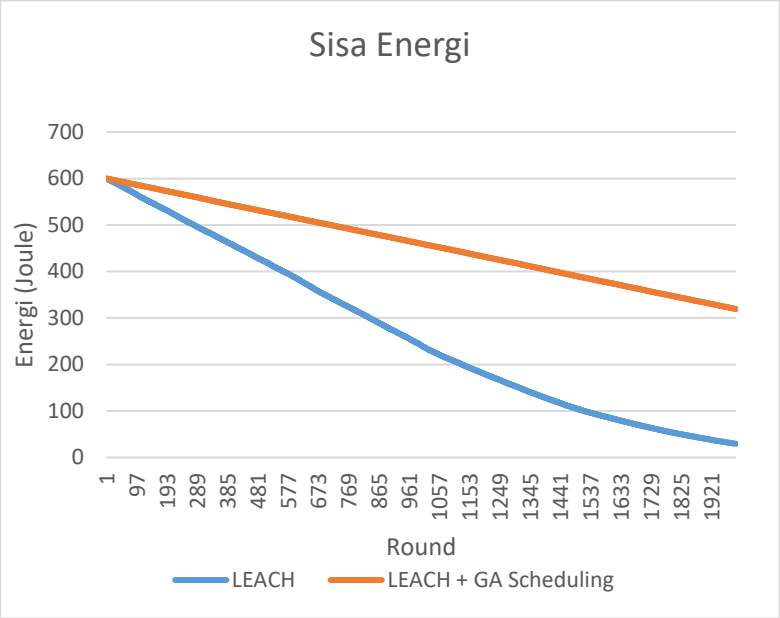


Gambar 5.3 Rata-rata Penggunaan Energi.

Berdasarkan hasil yang ditunjukkan pada gambar 5.2 terlihat perbedaan penggunaan energi yang cukup jauh pada WSN dengan *routing* LEACH dan GA *scheduling* dibandingkan WSN dengan *routing* LEACH tanpa *sleep scheduling*. Pada ronde ke-1 hingga ronde ke-1537, terlihat selisih penggunaan energi hingga 49%, namun pada sisa ronde akhir penggunaan energi terlihat sama. Hal tersebut terjadi karena jumlah *node* yang aktif pada WSN dengan *routing* LEACH tanpa *sleep scheduling* mengalami penurunan, secara langsung mempengaruhi penggunaan energi tiap ronde. Penggunaan energi pada WSN dengan *routing* LEACH dan GA *scheduling* cenderung lebih stabil hingga akhir ronde dibandingkan WSN dengan *routing* LEACH tanpa *sleep scheduling*. Perbedaan tersebut terjadi karena pada WSN dengan *routing* LEACH tanpa *sleep scheduling* semua *node* dalam keadaan aktif melakukan deteksi dan transmisi data.

Hal ini didukung dengan hasil rata-rata penggunaan energi pada gambar 5.3 dimana rata-rata penggunaan energi pada WSN dengan *routing* LEACH tanpa *sleep scheduling* sebesar 0,285527574 *joule* per ronde, jauh lebih tinggi dibandingkan

dengan rata-rata penggunaan energi pada WSN dengan *routing* LEACH dan GA *scheduling* sebesar 0,140418373 *joule* per ronde.



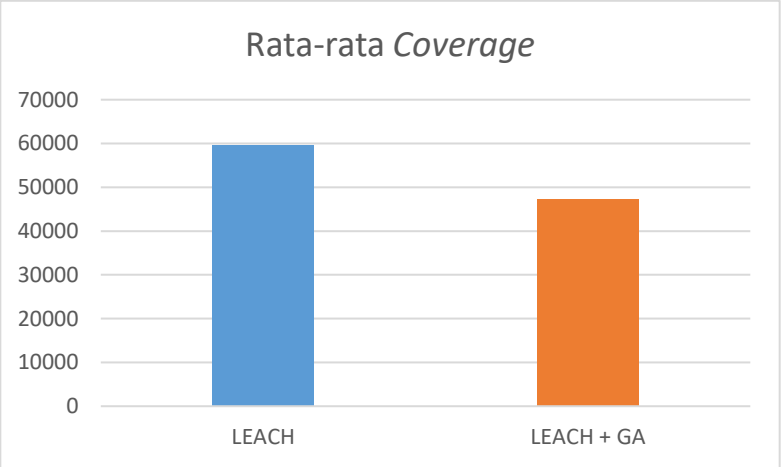
Gambar 5.4 Sisa Energi Pada Node

Berdasarkan hasil yang ditunjukkan pada gambar 5.4, terjadi penurunan yang berbeda pada dua kurva. Pada WSN dengan *routing* LEACH dan GA *scheduling* menunjukkan kurva menurun yang stabil, yang mencerminkan penurunan sisa energi yang stabil. Hal tersebut terjadi karena penggunaan energi pada tiap ronde yang relatif stabil. Detil penggunaan energi telah ditunjukkan pada gambar 5.2.

Sedangkan pada WSN dengan *routing* LEACH tanpa *sleep scheduling* menunjukkan kurva turun yang awalnya tajam menjadi landai. Penurunan kurva yang tajam mencerminkan sisa energi tiap ronde yang turun cukup banyak, dan menjadi landai yang menunjukkan penurunan sisa energi yang tidak banyak. Hal

tersebut bisa terjadi karena penggunaan energi pada tiap ronde yang sangat besar ketika jumlah *node* yang aktif masih banyak, dan kemudian di akhir mengalami penurunan penggunaan energi karena jumlah *node* yang aktif sudah berkurang. Detil penggunaan energi telah ditunjukkan pada gambar 5.2. Dengan demikian dapat disimpulkan bahwa dengan menerapkan GA *scheduling* dapat menghemat penggunaan energi hingga 49% sehingga sisa energi hingga akhir ronde lebih banyak.

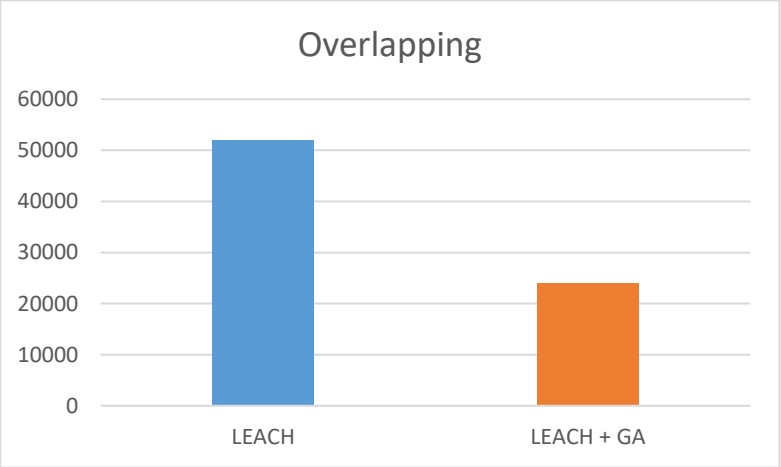
Dari pembahasan dalam segi *node* dan energi, dapat dilihat terdapat hubungan antara konsumsi energi dan jumlah *node*, dimana semakin sedikit energi yang digunakan pada tiap ronde maka jumlah *node* semakin banyak dan masa hidup *node* menjadi lebih lama.



Gambar 5.5 Rata-rata *Coverage*

Pada gambar 5.5 menunjukkan perbedaan rata-rata *coverage* yang tidak begitu signifikan dari WSN dengan *routing* LEACH tanpa *sleep scheduling* dan WSN dengan *routing* LEACH dengan GA *scheduling*. WSN dengan *routing* LEACH dan GA *scheduling*, rata-rata *coverage* mencapai 47308 *grid pixel*. Nilai *coverage* ini mengalami penurunan hingga 21% jika dibandingkan

dengan rata-rata *coverage* WSN dengan *routing* LEACH tanpa *sleep scheduling* yang mencapai 59536 *grid pixel*. Penurunan tersebut terjadi karena jumlah *node* yang aktif pada pengujian WSN dengan *routing* LEACH dengan GA *scheduling* lebih sedikit. Proses ubah *state* dari *active* menjadi *sleep* menyebabkan ada beberapa wilayah dalam area yang awalnya masuk menjadi tidak masuk dalam jangkauan deteksi.

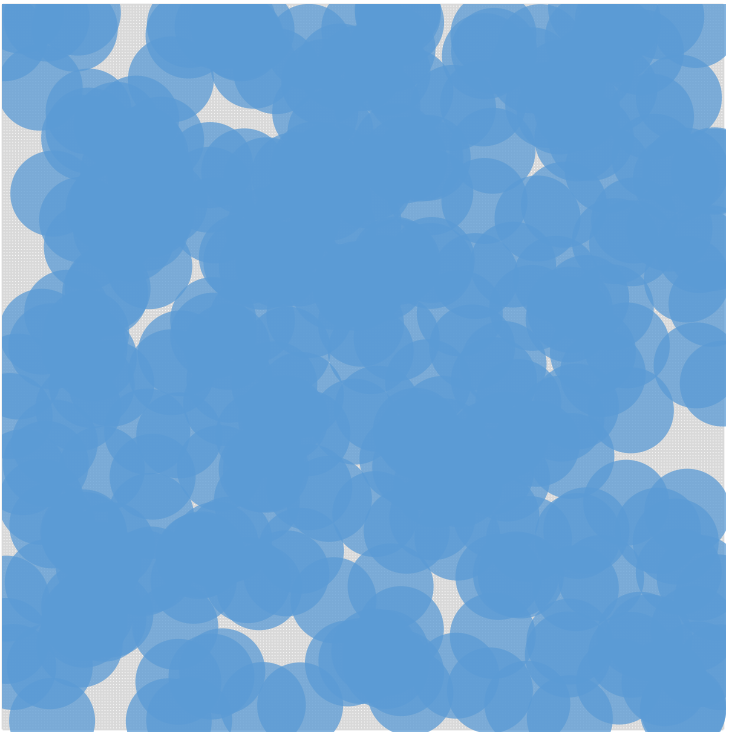


Gambar 5.6 Rata-rata *Overlapping*

Pada gambar 5.6 menunjukkan perbedaan rata-rata *overlapping* yang cukup jauh dari WSN dengan *routing* LEACH tanpa *sleep scheduling* dan WSN dengan *routing* LEACH dan GA *scheduling*. WSN dengan *routing* LEACH dan GA *scheduling* memiliki *overlapping* hingga 23924 *grid pixel*, secara presentase jauh lebih rendah 54% dibandingkan dengan WSN dengan *routing* LEACH tanpa *sleep scheduling* yang memiliki *overlapping* sebanyak 52003 *grid pixel*.

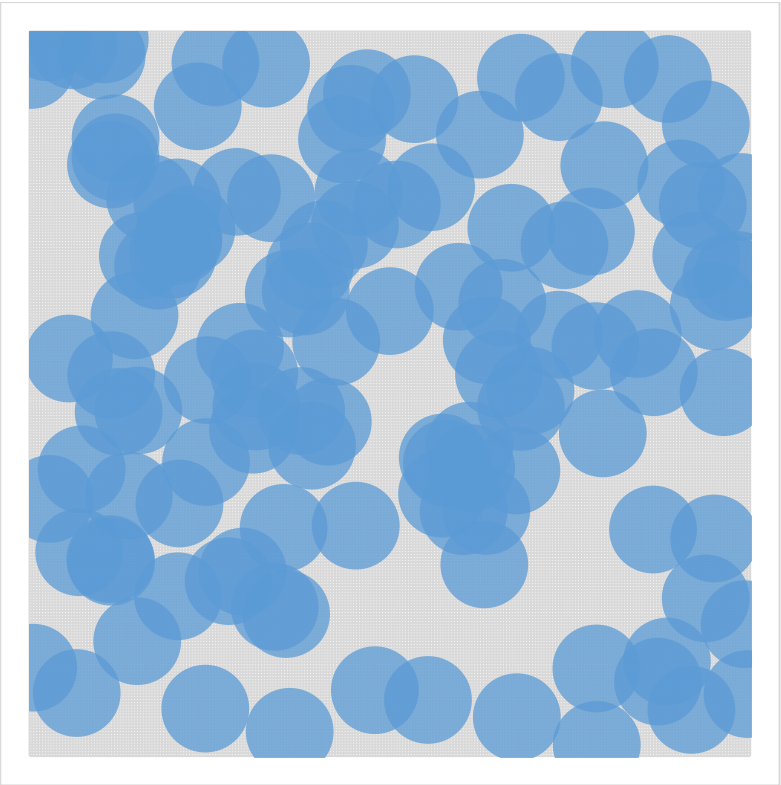
Perbedaan yang lumayan jauh tersebut dapat terjadi karena pada WSN dengan *routing* LEACH dan GA *scheduling* ada beberapa *node* yang diubah menjadi *sleep*, hal itu dapat mengoptimalkan beberapa *node* yang memiliki jangkauan deteksi

yang tumpang tindih menjadi hanya satu *node* saja yang mendeteksi area tersebut. Selain menghemat energi, hal ini juga dapat membuat proses deteksi menjadi lebih efisien.



Gambar 5.7 Hasil Plot *Coverage* sebelum *Sleep Scheduling.*

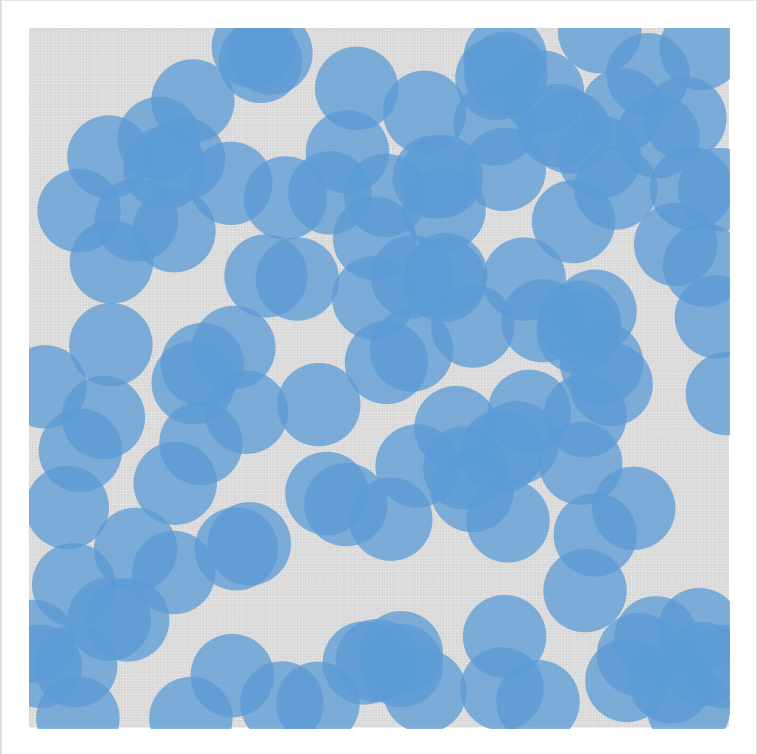
Pada gambar 5.7 terlihat hasil plot *coverage* sebelum *sleep scheduling* yang menunjukkan bahwa *coverage* hampir menutupi semua area, namun *overlapping* yang terbentuk terbilang besar. *Overlapping* yang besar terjadi karena banyak *node* yang berada pada posisi yang berdekatan, menyebabkan jangkauan deteksi menjadi tumpang tindih. Hal tersebut menunjukkan persebaran deteksi yang tidak efisien.



Gambar 5.8 Hasil Plot *Coverage* setelah *Sleep Scheduling* pada ronde 1.

Pada gambar 5.8 terlihat plot *coverage* setelah *sleep scheduling* yang menunjukkan bahwa terjadi pengurangan luas deteksi yang cukup signifikan. Pengurangan tersebut terjadi karena banyak *node* yang diubah menjadi *sleep*, yang menyebabkan beberapa bagian dalam area tidak dapat terdeteksi. Tidak efisiennya penentuan *sleep scheduling* karena pada awal dimulainya *sleep scheduling*, energi pada semua *node* berada pada keadaan penuh sehingga berpengaruh dalam penentuan nilai *fitness*. Namun jika dilihat pada segi *overlapping*, pengurangan yang terjadi sangat jauh. Hanya ada beberapa bagian wilayah yang

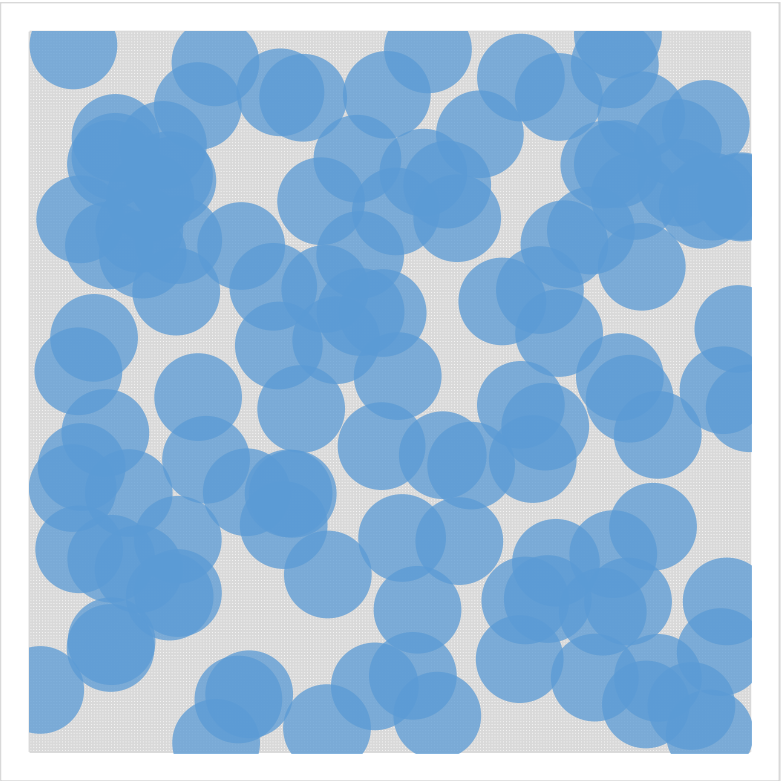
overlapping. Nilai *coverage* pada ronde 1 mencapai 44403 *grid pixel* dan *overlapping* mencapai 23265 *grid pixel*.



Gambar 5.9 Hasil Plot *Coverage* setelah *Sleep Scheduling* pada ronde 2.

Pada gambar 5.9 terlihat plot *coverage* setelah *sleep scheduling* pada ronde 2 yang menunjukkan konfigurasi *sleep scheduling* yang lebih baik dibandingkan dengan konfigurasi pada gambar 5.8. *Coverage* keseluruhan tetap mengalami penuruan namun dalam persebaran lebih merata dan *overlapping* lebih berkurang. Hal ini dapat terjadi karena beberapa *node* yang diubah menjadi *sleep* pada ronde sebelumnya berpotensi lebih untuk tetap aktif. Maka dengan demikian persebaran jangkauan lebih merata

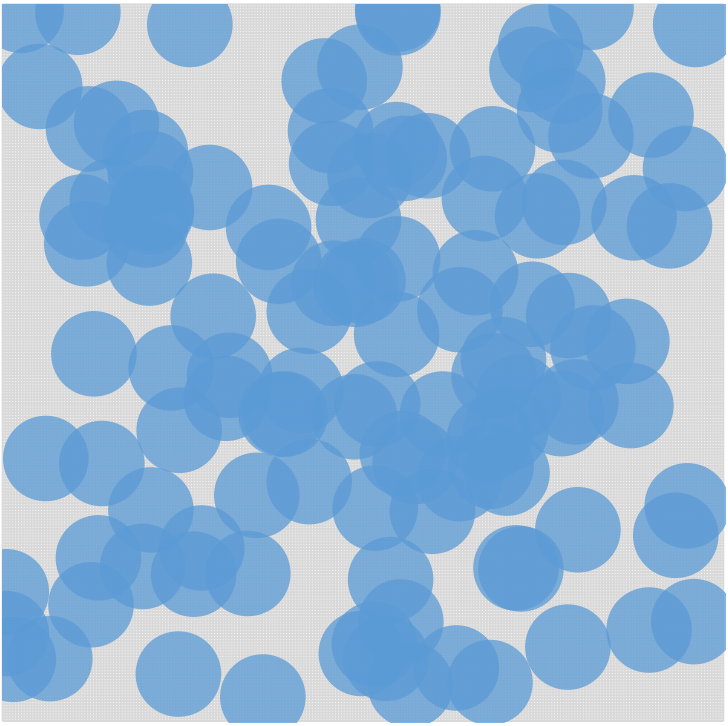
dan jangkauan deteksi menjadi lebih luas. Nilai *coverage* pada ronde 2 mencapai 45552 *grid pixel* dan *overlapping* mencapai 22293 *grid pixel*.



Gambar 5.10 Hasil Plot *Coverage* setelah *Sleep Scheduling* pada ronde 3.

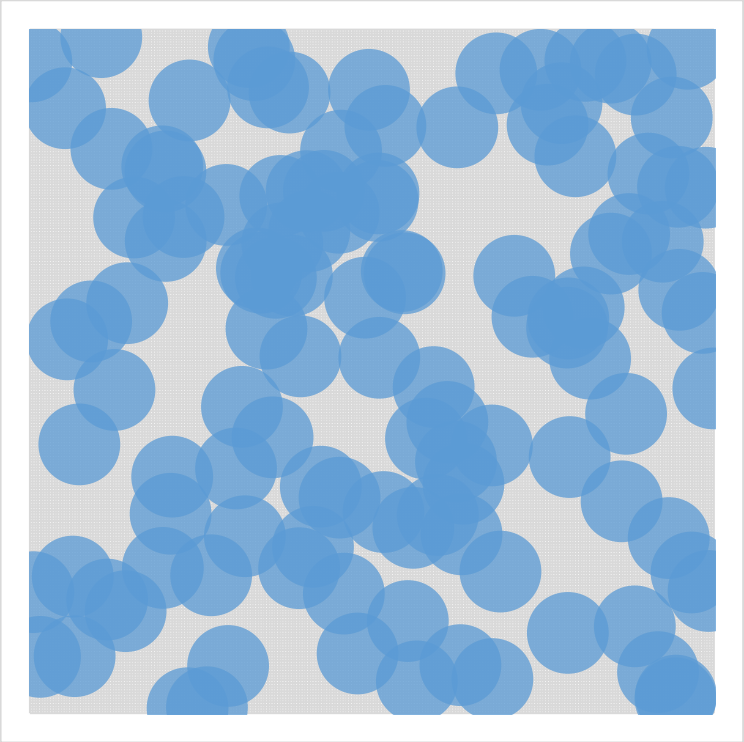
Pada gambar 5.10 terlihat plot *coverage* setelah penerapan *sleep scheduling* pada ronde 3 yang menunjukkan hasil yang berbeda dibandingkan konfigurasi pada gambar 5.9. Terjadi perbaikan *coverage* pada bagian kecil yang belum terdeteksi, adanya efisiensi pada area bagian pojok kiri bawah yang cukup diawasi oleh 1 *node* saja, dan pada area bagian tengah bawah yang

sudah diawasi oleh beberapa *node*. Dalam segi *overlapping* masih terjadi penumpukan dimana pada area bagian pojok kiri atas dan pojok kanan atas terdapat penumpukan jangkauan deteksi dan *overlapping* yang cukup luas. Meskipun begitu, luas *coverage* pada ronde ini mencapai 51074 *grid pixel*, lebih tinggi dari rata-rata *coverage* yaitu 47308. Dari segi *overlapping* mencapai 24250 *grid pixel*, termasuk di atas rata-rata namun dengan selisih sangat sedikit dengan nilai rata-rata *overlapping* yaitu 23923 *grid pixel*.



Gambar 5.11 Hasil Plot *Coverage* setelah *Sleep Scheduling* pada ronde 4

Pada gambar 5.11 terlihat plot *coverage* setelah *sleep scheduling* pada ronde 4 yang menunjukkan perubahan yang cukup sigifikan. Pada bagian tengah atas terdapat area yang sebelumnya pada ronde 2 dan 3 ada *node* yang aktif untuk mendeteksi, sekarang sudah tidak ada. Terlihat juga *overlapping* terjadi di bagian tengah, dimana pada ronde 3 tidak terjadi *overlapping*. Hal ini dapat terjadi karena terdapat hubungan dengan ronde sebelumnya, dimana *node* yang aktif pada ronde-ronde sebelumnya memiliki kemungkinan tinggi untuk diubah menjadi *sleep*. Begitupun dengan *node* yang sebelumnya berada pada keadaan *sleep* lebih mungkin untuk diaktifkan pada ronde selanjutnya.



Gambar 5.12 Hasil Plot *Coverage* setelah *Sleep Scheduling* pada ronde 5

Pada gambar 5.12, terlihat plot *coverage* setelah *sleep scheduling* pada ronde 5 yang menunjukkan penurunan jika dibandingkan dengan hasil sebelumnya. Ada beberapa bagian area yang sebelumnya dapat dijangkau oleh beberapa *node*, sekarang tidak ada yang mendeteksi bagian tersebut. Seperti pada penjelasan 5.11 bahwa *node* yang sebelumnya telah aktif memiliki kemungkinan lebih besar untuk diubah menjadi *sleep*. *Overlapping* pada bagian tengah area yang terjadi pada ronde sebelumnya sudah tidak terlihat begitu luas.

Dari data yang telah ditunjukkan terdapat hubungan antara *coverage*, *overlapping*, dan *node* yang aktif. Semakin banyak *node* yang aktif dapat memperlebar *coverage*, namun juga memperlebar *overlapping*. Dan tak bisa dihindari bahwa penerapan *sleep scheduling* tentu akan mengurangi jumlah *node* yang aktif, dan tentu berbanding lurus dengan penurunan *coverage* keseluruhan pada WSN.

5.4.2 Hasil Skenario Uji Coba 2

Berikut hasil skenario uji coba antara variasi jumlah node 50, 150, dan 300:

Tabel 5.6 Hasil Pengujian Skenario 2

	50	150	300
Rata-rata penggunaan energi	0,03668	0,092837	0,140418
Coverage sebelum *sleep scheduling*	26385	49562	59536
Coverage setelah *sleep scheduling*	17328	38207	47308
Luas *overlapping*	3861	28535	52003

area sebelum *sleep scheduling*			
Luas *overlapping area* setelah *sleep scheduling*	1555	11037	23923

Dari tabel 5.6 menunjukkan hasil uji coba pada variasi jumlah *node*. Pada pengujian dengan variasi 50 *node*, terjadi penurunan *coverage* setelah penerapan *sleep scheduling* yang cukup besar yaitu sebesar 34%, sedangkan dari segi *overlapping* mengalami penurunan hingga 60%. Pada variasi 150 *node*, segi *coverage* mengalami penurunan hingga 23% dan segi *overlapping* mengalami penurunan hingga 61%. Melihat pada variasi 300 *node*, terjadi penurunan *coverage* sebesar 21% dan penurunan *overlapping* sebesar 54%. Dari hasil tersebut dapat terlihat ada pengaruh jumlah *node* terhadap performa GA *scheduling*.

Dari segi *coverage*, penurunan paling banyak terjadi pada variasi 50 *node* yang mencapai 34%, dimana itu adalah variasi jumlah *node* paling terkecil. Dapat diamati juga selisih penuruhan *coverage* pada variasi 50 *node* dan 150 *node* yang lumayan signifikan. Sementara selisih penuruhan *coverage* antara variasi 150 *node* dan 300 *node* sangat sedikit.

Dari segi *overlapping*, terlihat penurunan *overlapping* paling sedikit pada variasi 300 *node*, dan terbesar pada variasi 150 *node*. Selisih penurnan antara variasi 50 *node* dan 150 *node* sangat sedikit, namun selisih penurunan *overlapping* pada 150 *node* dan 300 *node* justru lebih jauh jika dibandingkan dengan selisih penurunan *overlapping* pada variasi 50 *node* dan 150 *node*. Meskipun begitu, perbedaan penurunan *overlapping* bisa dibilang stabil karena tidak memiliki selisih yang sangat jauh.

Dari data yang ada, dapat disimpulkan bahwa semakin sedikit *node* yang berjalan pada WSN dengan *routing* LEACH dan

GA *scheduling* maka penurunan *coverage* semakin besar. Namun jika semakin banyak *node* maka penurunan *coverage* semakin kecil. Namun dari segi *overlapping* tidak menampakkan perbedaan yang signifikan.

5.4.3 Hasil Skenario Uji Coba 3

Berikut hasil skenario uji coba antara variasi luas area 100x100, 150x150, dan 250x250:

Tabel 5.7 Hasil Pengujian Skenario 3

	100 x 100	150 x 150	250 x 250
Rata-rata penggunaan energi	0,034174	0,043605476	0,0518238
Coverage sebelum *sleep scheduling*	9892	20398	38160
Coverage setelah *sleep scheduling*	9425	17782	23189
Luas *overlapping* area sebelum *sleep scheduling*	9345	16587	18251
Luas *overlapping area* setelah *sleep scheduling*	7002	8503	3205

Dari tabel 5.7 menunjukkan hasil uji coba pada variasi luas area. Pada area pengujian yang kecil yaitu 100x100, terjadi penurunan *coverage* yang sangat kecil yaitu sebesar 5%, dan terjadi

penurunan *overlapping* hingga 25%. Pada area 150x150, terjadi penurunan *coverage* sebesar 13% dan terjadi penurunan *overlapping* hingga 49%. Pada area pengujian yang paling besar yaitu 250x250, terjadi penurunan *coverage* hingga 39% dan terjadi penurunan *overlapping* hingga 82%. Dalam pengujian ini terlihat bahwa ada hubungan antara perbedaan luas area dan performa GA *scheduling*.

Jika ketiga variasi area dibandingkan, terlihat bahwa pada luas area terkecil yaitu 100x100, selisih penurunan *coverage* terjadi paling kecil jika dibandingkan dengan luas area 150x150 dan 250x250. Dapat diamati juga ketika melihat pada variasi area 150x150 terjadi penurunan *coverage* yang lebih besar jika dibandingkan dengan variasi area 100x100, begitu juga jika melihat pada variasi area 250x250 yang memiliki penurunan *coverage* paling besar jika dibandingkan dengan variasi area 100x100 dan 150x150.

Penurunan *overlapping* paling besar terjadi pada variasi luas area paling besar yaitu 250x250, dan terkecil pada variasi luas area 100x100. Dapat diamati juga pada variasi area 100x100 dan 150x150 penurunan *overlapping* bertambah dari 25% menjadi 49%. Dan pada variasi area 150x150 dan 250x250 penurunan *overlapping* bertambah signifikan dari 49% menjadi 82%.

Dari data yang ada, dapat diambil kesimpulan bahwa luas area yang kecil pada WSN dengan *routing* LEACH dan GA *scheduling* dapat berpengaruh pada *coverage* dan *overlapping*, dimana semakin kecil area deteksi maka penurunan *coverage* semakin kecil, dan semakin besar area deteksi maka penurunan *overlapping* semakin besar.

(Halaman ini sengaja dikosongkan)

BAB VI
KESIMPULAN DAN SARAN

Bab ini membahas mengenai kesimpulan yang diperoleh dari tugas akhir yang telah dikerjakan dan saran terkait pengembangan dari tugas akhir ini yang dapat dilakukan pada masa yang akan datang.

6.1 Kesimpulan

Kesimpulan yang diperoleh dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

1. *Sleep scheduling* pada *wireless sensor network* bekerja dengan menghasilkan konfigurasi yang menentukan status pada tiap-tiap *node*.
2. *Genetic algorithm* mengoptimasi hasil *sleep scheduling* agar hasil konfigurasi tidak hanya menghemat penggunaan energi namun juga dapat mengurangi terjadinya *overlapping* dan menjaga agar *coverage* tidak mengalami penurunan yang cukup jauh.
3. Tujuan dari penerapan *sleep scheduling* menggunakan *genetic algorithm* adalah mengurangi penggunaan energi, mengurangi *overlapping area*, namun tidak mempengaruhi *coverage* jaringan secara signifikan. Hal ini ditunjukkan pada hasil uji coba dimana penggunaan energi pada WSN dengan *sleep scheduling* menggunakan *genetic algorithm* dapat menghemat konsumsi energi hingga 49%. Selain itu juga dari segi *overlapping* area mengalami penurunan hingga 82%. Meskipun begitu, *coverage* pada lingkungan WSN dengan *sleep scheduling* menggunakan *genetic algorithm* mengalami penurunan hingga 34%. Penurunan *coverage* dan *overlapping* dapat dipengaruhi oleh jumlah *node* yang berjalan pada *wireless sensor network* dan luas area yang dipantau.

6.2 Saran

Saran yang diberikan dari hasil uji coba dan evaluasi pada tugas akhir ini adalah sebagai berikut:

1. Menyesuaikan bobot kriteria dalam penghitungan *fitness* pada *genetic algorithm* yang dapat meningkatkan akurasi.
2. Menambahkan nilai pembobotan pada *overlapping region* sesuai dengan luas tiap *overlapping region.*

DAFTAR PUSTAKA

[1] A. Svetoslav, "An overview of wireless communication technologies used in wireless sensor networks," *International Scientific Conference eRA-8,* 2013.

[2] P. Rawat, K. D. Singh, H. Chaouchi and J. M. Bonnin, "Wireless sensor networks: A survey on recent developments and potential synergies," *The Journal of supercomputing 68.1,* pp. 1-48, 2014.

[3] B. Zebbane, M. Chenait and N. Badache, "Energy-Efficient Protocol based Sleep-Scheduling for Wireless Sensor Network," *2012 IEEE International Conference on Complex Systems (ICCS),* pp. 1-6, 2012.

[4] L. Haldurai, T. Madhubala and R. Rajalakshmi, "A Study on Genetic Algorithm and its Applications," *International Journal of Computer Sciences and Engineering,* vol. 4, no. 10, pp. 139-143, 2016.

[5] "PyCharm: the Python IDE for Professional Developers by JetBrains," 18 June 2020. [Online]. Available: https://www.jetbrains.com/pycharm/.

[6] A. Singh, S. Rathkanthiwar and S. Kakde, "LEACH Based-Energy Efficient Routing Protocol for Wireless Sensor Network," *International Conference on Electrical, Electronics, and Optimization Techniques (ICEEOT),* pp. 4654-4658, 2016.

[7] Q. Liao and H. Zhu, "An Energy Balanced Clustering Algorithm Based on LEACH Protocol," *Applied Mechanics and Materials,* Vols. 341-342, pp. 1138-1143, 2013.

[8] "BeginnersGuide Overview," 19 June 2020. [Online]. Available: https://wiki.python.org/moin/BeginnersGuide/Overview.

[9] C. Yu, W. Guo and G. Chen, "Energy-balanced Sleep Scheduling Based on Particle Swarm Optimization in Wireless Sensor Network," *2012 IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum*, pp. 1249-1255, 2012.

[10] M. Karthikadevi and S. Pavalarajan, "Sleep Scheduling Strategies In Wireless Sensor Network," *Advances in Natural and Applied Sciences*, pp. 635-641, 2017.

[11] "Simplified Wrapper and Interface Generator". [Online]. Available: http://www.swig.org/. [Accessed: 22 July 2020].

[12] "The C++ Resources Network". [Online]. Available: http://www.cplusplus.com/. [Accessed: 22 July 2020].

LAMPIRAN

KODE SUMBER

1. Kode sumber grid.py

```
1.      import numpy as np
2.      import logging
3.      import config as cf
4.  from python.utils.utils import *
5.      from python.utils.region import *
6.
7.      def _adjust2grid(pos):
8.        return
cf.GRID_PRECISION*int(pos/cf.GRID_PRECISION)
9.
10.     class Grid(object):
11.
12.       def __init__(self):
13.         self._pixels = {}
14.         self._exclusive_regions = []
15.
16.       def _paint_pixel(self, x, y, id):
17.         if x not in self._pixels: # add line if it does
not exist
18.           self._pixels[x] = {}
19.
20.         if y in self._pixels[x]:
21.           owners = self._pixels[x][y]
22.           owners.append(id)
23.           logging.debug("overlapping pixel %s %s" %(x,
y))
24.         else:
25.           self._pixels[x][y] = [id]
26.           logging.debug("painting pixel %s %s" %(x, y))
27.
28.       def add_node(self, node, coverage_radius):
29.         logging.info("adding node %d to grid" %
(node.id))
30.         initial_x = _adjust2grid(node.pos_x -
coverage_radius)
```

```
31.     initial_y = _adjust2grid(node.pos_y -
 coverage_radius)
32.     final_x   = _adjust2grid(node.pos_x +
 coverage_radius)
33.     final_y   = _adjust2grid(node.pos_y +
 coverage_radius)
34.     if initial_x < 0.0:
35.       initial_x = 0.0
36.     if initial_y < 0.0:
37.       initial_y = 0.0
38.     if final_x > _adjust2grid(cf.AREA_WIDTH):
39.       final_x   = _adjust2grid(cf.AREA_WIDTH)
40.     if final_y > _adjust2grid(cf.AREA_LENGTH):
41.       final_y   = _adjust2grid(cf.AREA_LENGTH)
42.
43.     for pixel_x in np.arange(initial_x, final_x,
 cf.GRID_PRECISION):
44.       for pixel_y in np.arange(initial_y, final_y,
 cf.GRID_PRECISION):
45.         distance =
 calculate_distance_point(pixel_x, pixel_y,
46.
 node.pos_x, node.pos_y)
47.         if distance < coverage_radius:
48.
 self._paint_pixel(str(_adjust2grid(pixel_x)),
49.
 str(_adjust2grid(pixel_y)),
50.                             node.id)
```

2. Kode Sumber region.py

```
1.  class Region(object):
2.    def __init__(self, area, owners=set()):
3.      self.area   = area
4.      self.owners = owners
5.
6.    def __str__(self):
7.      to_print = ""
8.      for owner in self.owners:
9.        to_print += " " + str(owner)
10.     return to_print + " " + str(self.area)
```

3. Kode Sumber region_converter.py

```
class RegionsConverter(list):

  _area_single_pixel = cf.GRID_PRECISION**2
  _exclude_area_if = 0.005

  def __init__(self, grid):
    logging.info('Creating Regions instance.')
    self.extend(grid._exclusive_regions)
    self._grid2regions(grid._pixels)
    self._extract_exclusive_regions()
    logging.info(self)

  def _grid2regions(self, pixels):
    logging.info('converting grid to regions.')
    for x, line in pixels.items():
      for y, pixel in line.items():
        owners = set(pixels[x][y])
        region = self._get_region(owners)
        if region:
          region.area += self._area_single_pixel
        else:
          new_region = Region(self._area_single_pixel, owners)
          self.append(new_region)

  def _extract_exclusive_regions(self):
    logging.info('extracting exclusive regions.')
    self._exclusive_regions = {}
    del_idx = []
    for idx, region in enumerate(self):
      if len(region.owners) == 1:
        owner = list(region.owners)[0]
        self._exclusive_regions[owner] = region.area
        del_idx.append(idx)

    for idx in del_idx[::-1]:
        del self[idx]
```

```
37.
38.   def _remove_small_regions(self):
39.     logging.info('removing small regions.')
40.     total_coverage = self._get_total_coverage()
41.     #print("total coverage %f" %(total_coverage))
42.     del_idx = []
43.     for idx, region in enumerate(self):
44.       percentage = region.area/total_coverage
45.       if percentage < self._exclude_area_if:
46.         del_idx.append(idx)
47.
48.     for idx in del_idx[::-1]:
49.         del self[idx]
50.
51.   def _get_region(self, owners):
52.     for region in self:
53.       if owners == region.owners:
54.         return region
55.
56.     return 0
57.
58.   def __str__(self):
59.     sum = 0.0
60.     regions_str = ''
61.     for owner, area in
    self._exclusive_regions.items():
62.       regions_str += "ex: %s, %f \n" %(str(owner),
    area)
63.       sum += area
64.     for region in self:
65.       regions_str += "ov: %s, %f \n"
    %(str(region.owners), region.area)
66.       sum += region.area
67.     regions_str += "total area: %f\n" %(sum)
68.     return regions_str
69.
70.   def _get_total_coverage(self):
71.     """Sums the areas from every region."""
72.     coverage = 0.0
73.     if hasattr(self, '_exclusive_regions'):
74.       for area in self._exclusive_regions:
75.         coverage += area
76.     for region in self:
```

```
77.       coverage += region.area
78.
79.     return coverage
80.
81.   def _check(self, exclusive, overlapping):
82.     for owner, area in exclusive.items():
83.       assert area >= 0.0, "Negative region found!"
84.     for region in self:
85.       assert region.area >= 0.0, "Negative region
    found!"
86.
87.   def convert(self):
88.     overlapping_regions = []
89.     for region in self:
90.
    overlapping_regions.append((list(region.owners),
    region.area))
91.
92.     self._check(self._exclusive_regions,
    overlapping_regions)
93.     return self._exclusive_regions,
    overlapping_regions
```

4. Kode Sumber node.py

```
1.  import config as cf
2.  import numpy as np
3.  from python.network.energy_source import *
4.  from python.utils.utils import *
5.
6.  class Node(object):
7.    def __init__(self, id, parent = None):
8.      self.pos_x = np.random.uniform(0,
    cf.AREA_WIDTH)
9.      self.pos_y = np.random.uniform(0,
    cf.AREA_LENGTH)
10.
11.     if id == cf.BSID:
12.       self.energy_source = PluggedIn(self)
```

```
13.     else:
14.       self.energy_source = Battery(self)
15.
16.     self.id = id
17.     self.network_handler = parent
18.
19.     self.reactivate()
20.
21.   def reactivate(self):
22.     self.alive = 1
23.     self.tx_queue_size = 0
24.     self._next_hop = cf.BSID
25.     self.distance_to_endpoint = 0
26.     self.amount_sensed = 0
27.     self.amount_transmitted = 0
28.     self.amount_received = 0
29.     self.membership = cf.BSID
30.     # aggregation function determines the cost of
    forwarding messages
31.     # (in number of bits)
32.     self.aggregation_function = lambda x: 0
33.     self.time_of_death = cf.INFINITY
34.     self._is_sleeping = 0
35.     self.sleep_prob = 0.0
36.     # for coverage purposes
37.     self.neighbors = []
38.     self.nb_neighbors = -1
39.     self.exclusive_radius = 0
40.
41.   @property
42.   def next_hop(self):
43.     return self._next_hop
44.
45.   @next_hop.setter
46.   def next_hop(self, value):
47.     self._next_hop = value
48.     distance = calculate_distance(self,
    self.network_handler[value])
49.     self.distance_to_endpoint = distance
50.
51.   @property
52.   def is_sleeping(self):
53.     # if self.is_head():
```

```
    #   self._is_sleeping = 0
    return self._is_sleeping

  @is_sleeping.setter
  def is_sleeping(self, value):
    self._is_sleeping = value

  def _only_active_nodes(func):
    def wrapper(self, *args, **kwargs):
      if self.alive and not self.is_sleeping:
        func(self, *args, **kwargs)
        return 1
      else:
        return 0
    return wrapper

  @_only_active_nodes
  def update_sleep_prob(self):
    if self.next_hop == cf.BSID:
      self.sleep_prob = 0.0
    else:
      self.sleep_prob = 0.5

    return

  def is_head(self):
    if self.next_hop == cf.BSID and self.id !=
cf.BSID and self.alive:
      return 1
    return 0

  def is_ordinary(self):
    return 1 if self.next_hop != cf.BSID and
self.id != cf.BSID else 0

  @_only_active_nodes
  def _aggregate(self, msg_length):
    logging.debug("node %d aggregating." %
(self.id))
    aggregation_cost =
self.aggregation_function(msg_length)
    self.tx_queue_size += aggregation_cost

```

```
93.     energy = cf.E_DA * aggregation_cost
94.     self.energy_source.consume(energy)
95.
96.   @_only_active_nodes
97.   def transmit(self, msg_length=None,
    destination=None):
98.     logging.debug("node %d transmitting." %
    (self.id))
99.     if not msg_length:
100.            msg_length = self.tx_queue_size
101.          msg_length += cf.HEADER_LENGTH
102.
103.          if not destination:
104.            destination =
    self.network_handler[self.next_hop]
105.            distance = self.distance_to_endpoint
106.          else:
107.            distance = calculate_distance(self,
    destination)
108.
109.          energy = cf.E_ELEC
110.          if distance > cf.THRESHOLD_DIST:
111.            energy += cf.E_MP * (distance**4)
112.          else:
113.            energy += cf.E_FS * (distance**2)
114.          energy *= msg_length
115.
116.          self.tx_queue_size = 0
117.          self.amount_transmitted += msg_length
118.
119.          self.energy_source.consume(energy)
120.
121.        @_only_active_nodes
122.        def receive(self, msg_length):
123.          logging.debug("node %d receiving." %
    (self.id))
124.          self._aggregate(msg_length -
    cf.HEADER_LENGTH)
125.
126.          self.amount_received += msg_length
127.
128.          energy = cf.E_ELEC * msg_length
129.          self.energy_source.consume(energy)
```

```
130.
131.        @_only_active_nodes
132.        def sense(self):
133.          self.tx_queue_size = cf.MSG_LENGTH
134.          self.amount_sensed += cf.MSG_LENGTH
135.
136.        def battery_depletion(self):
137.          self.alive = 0
138.          self.sleep_prob = 0.0
139.          self.time_of_death =
    self.network_handler.round
140.          self.network_handler.deaths_this_round
    += 1
```

5. Kode Sumber network.py

```
1.  class Network(list):
2.    def __init__(self, init_nodes=None):
3.      logging.debug('Instantiating nodes...')
4.      if init_nodes:
5.        self.extend(init_nodes)
6.      else:
7.        nodes = [Node(i, self) for i in range(0,
    cf.NB_NODES)]
8.        self.extend(nodes)
9.        # last node in nodes is the base station
10.       base_station = Node(cf.BSID, self)
11.       base_station.pos_x = cf.BS_POS_X
12.       base_station.pos_y = cf.BS_POS_Y
13.       self.append(base_station)
14.
15.     self._dict = {}
16.     for node in self:
17.       self._dict[node.id] = node
18.
19.     self.perform_two_level_comm = 1
20.     self.round = 0
21.     self.centroids = []
22.     self.routing_protocol = None
```

```
23.     self.sleep_scheduler_class = None
24.
25.     self.initial_energy =
    self.get_remaining_energy()
26.     self.first_depletion = 0
27.     self.per50_depletion = 0
28.     self.energy_spent = []
29.
30.   def reset(self):
31.     for node in self:
32.       node.energy_source.recharge()
33.       node.reactivate()
34.
35.     self[-1].pos_x = cf.BS_POS_X
36.     self[-1].pos_y = cf.BS_POS_Y
37.
38.     self.round = 0
39.     self.centroids = []
40.     self.energy_spent = []
41.
42.     self.routing_protocol = None
43.     self.sleep_scheduler_class = None
44.
45.     self.first_depletion = 0
46.     self.per50_depletion = 0
47.     self.perform_two_level_comm = 1
48.
49.   def simulate(self):
50.     tracer = Tracer()
51.
52.     self.routing_protocol.pre_communication(self)
53.
54.     all_alive = 1
55.     percent50_alive = 1
56.     self.deaths_this_round = 0
57.
58.     if self.sleep_scheduler_class:
59.       self._sleep_scheduler = SleepScheduler(self,
    self.sleep_scheduler_class)
60.
61.     for round_nb in range(0, cf.MAX_ROUNDS):
62.       self.round = round_nb
```

```
63.       print_args = (round_nb,
    self.get_remaining_energy())
64.       print("round %d: total remaining energy: %f"
    % print_args)
65.       nb_alive_nodes = self.count_alive_nodes()
66.       if nb_alive_nodes == 0:
67.         break
68.
    tracer['alive_nodes'][2].append(nb_alive_nodes)
69.
    tracer['energies'][2].append(self.get_remaining_ene
    rgy())
70.
71.       if self.sleep_scheduler_class:
72.         log = self._sleep_scheduler.schedule()
73.         for key, value in log.items():
74.           tracer[key][2].append(value)
75.
76.       self.routing_protocol.setup_phase(self,
    round_nb)
77.
78.       if self.deaths_this_round != 0:
79.         if all_alive == 1:
80.           all_alive = 0
81.           self.first_depletion = round_nb
82.         if float(nb_alive_nodes)/float(cf.NB_NODES)
    < 0.5 and percent50_alive == 1:
83.           percent50_alive = 0
84.           self.per50_depletion = round_nb
85.
86.       self.deaths_this_round = 0
87.       self.routing_protocol.broadcast(self)
88.
89.       self._run_round(round_nb)
90.
91.
    tracer['first_depletion'][2].append(self.first_depl
    etion)
92.
    tracer['50per_depletion'][2].append(self.per50_depl
    etion)
93.
94.     return tracer
```

```
95.
96.   def _run_round(self, round):
97.     before_energy = self.get_remaining_energy()
98.     for i in range(0, cf.MAX_TX_PER_ROUND):
99.       self._sensing_phase()
100.            self._communication_phase()
101.          after_energy =
    self.get_remaining_energy()
102.          self.energy_spent.append(before_energy -
    after_energy)
103.
104.        def _sensing_phase(self):
105.          for node in self.get_alive_nodes():
106.            node.sense()
107.
108.        def _communication_phase(self):
109.
110.          alive_nodes = self.get_alive_nodes()
111.          if self.perform_two_level_comm == 1:
112.            self._two_level_comm(alive_nodes)
113.          else:
114.            self._recursive_comm(alive_nodes)
115.
116.        def _recursive_comm(self, alive_nodes):
117.          """Hierarchical communication using
    recursivity. This method suppo-
118.          ses that there is no cycle in the
    network (network is a tree).
119.          Otherwise, expect infinite loop.
120.          """
121.          next_alive_nodes = alive_nodes[:]
122.          for node in alive_nodes:
123.            depends_on_other_node = 0
124.            for other_node in alive_nodes:
125.              if other_node.next_hop == node.id:
126.                depends_on_other_node = 1
127.                break
128.
129.            if not depends_on_other_node:
130.              node.transmit()
131.              next_alive_nodes = [n for n in
    next_alive_nodes if n != node]
132.
```

```
133.          if len(next_alive_nodes) == 0:
134.            return
135.          else:
136.            self._recursive_comm(next_alive_nodes)
137.
138.        def _two_level_comm(self, alive_nodes):
139.          for node in self.get_ordinary_nodes():
140.            node.transmit()
141.          for node in self.get_heads():
142.            node.transmit()
143.
144.        def get_alive_nodes(self):
145.          return [node for node in self[0:-1] if
    node.alive]
146.
147.        def get_active_nodes(self):
148.          is_active = lambda x: x.alive and not
    x.is_sleeping
149.          return [node for node in self[0:-1] if
    is_active(node)]
150.
151.        def get_ordinary_nodes(self):
152.          return [node for node in self if
    node.is_ordinary() and node.alive]
153.
154.        def get_heads(self, only_alives=1):
155.          input_set = self.get_alive_nodes() if
    only_alives else self
156.          return [node for node in input_set if
    node.is_head()]
157.
158.        def get_sensor_nodes(self):
159.          return [node for node in self[0:-1]]
160.
161.        def get_average_energy(self):
162.          return np.average(self.energy_spent)
163.
164.        def someone_alive(self):
165.          for node in self[0:-1]:
166.            if node.alive == 1:
167.              return 1
168.          return 0
169.
```

```
170.        def count_alive_nodes(self):
171.          return sum(x.alive for x in self[:-1])
172.
173.        def get_BS(self):
174.          return self[-1]
175.
176.        def get_node(self, id):
177.          return self._dict[id]
178.
179.        def notify_position(self):
180.          for node in self.get_alive_nodes():
181.
    node.transmit(msg_length=cf.MSG_LENGTH,
    destination=self.get_BS())
182.
183.        def broadcast_next_hop(self):
184.          base_station = self.get_BS()
185.          for node in self.get_alive_nodes():
186.
    base_station.transmit(msg_length=cf.MSG_LENGTH,
    destination=node)
187.
188.        def get_nodes_by_membership(self,
    membership, only_alives=1):
189.          input_set = self.get_alive_nodes() if
    only_alives else self
190.          condition = lambda node: node.membership
    == membership and node.id != cf.BSID
191.          return [node for node in input_set if
    condition(node)]
192.
193.        def get_remaining_energy(self,
    ignore_nodes=None):
194.          set = self.get_alive_nodes()
195.          if len(set) == 0:
196.            return 0
197.          if ignore_nodes:
198.            set = [node for node in set if node
    not in ignore_nodes]
199.          transform = lambda x:
    x.energy_source.energy
200.          energies = [transform(x) for x in set]
201.          return sum(x for x in energies)
```

```
202.
203.        def set_aggregation_function(self,
    function):
204.          for node in self:
205.            node.aggregation_function = function
206.
207.        def split_in_clusters(self,
    nb_clusters=cf.NB_CLUSTERS):
208.          clusters = []
209.          for cluster_idx in range(0,
    nb_clusters):
210.            nodes =
    self.get_nodes_by_membership(cluster_idx)
211.            cluster = Network(init_nodes=nodes)
212.            cluster.append(self.get_BS())
213.            clusters.append(cluster)
214.          return clusters
215.
216.        def _calculate_nb_neighbors(self,
    target_node):
217.          if target_node.nb_neighbors != -1:
218.            # only check if there are dead nodes
219.            all_neighbors = target_node.neighbors
220.            nb_dead_neighbors = sum(1 for x in
    all_neighbors if not x.alive)
221.            target_node.neighbors[:] = [x for x in
    all_neighbors if x.alive]
222.            return target_node.nb_neighbors -
    nb_dead_neighbors
223.
224.          nb_neighbors = 0
225.          shortest_distance = cf.COVERAGE_RADIUS*2
226.          for node in self.get_alive_nodes():
227.            if node == target_node:
228.              continue
229.            distance =
    calculate_distance(target_node, node)
230.            if distance <= cf.COVERAGE_RADIUS:
231.              nb_neighbors += 1
232.              target_node.neighbors.append(node)
233.              if distance < shortest_distance:
234.                shortest_distance = distance
235.
```

```
236.          if shortest_distance != cf.INFINITY:
237.            exclusive_radius = shortest_distance -
    cf.COVERAGE_RADIUS
238.            if exclusive_radius < 0:
239.              exclusive_radius = 0.0
240.
241.          node.nb_neighbors = nb_neighbors
242.          node.exclusive_radius = exclusive_radius
243.
244.        def update_neighbors(self):
245.          for node in self.get_alive_nodes():
246.            self._calculate_nb_neighbors(node)
247.
248.          self.update_sleep_prob()
249.
250.        def update_sleep_prob(self):
251.          for node in self.get_alive_nodes():
252.            node.update_sleep_prob()
```

BIODATA PENULIS

Chendra Sena Oemaryoga lahir di Kediri pada tanggal 10 Januari 1998. Penulis menempuh pendidikan formal di SDN 2 Kandangan (2004-2010), SMPN 2 Pare (2010-2013), SMAN 2 Pare (2013-2016), dan Teknik Informatika ITS Surabaya (2016-2020). Bidang studi yang diambil oleh penulis saat berkuliah di Departemen Teknik Informatika ITS adalah Komputasi Berbasis Jaringan (KBJ). Penulis aktif dalam organisasi Himpunan Mahasiswa Teknik Computer-Informatika tahun kepengurusan 2017-2018 dan 2018-2019. Penulis juga aktif dalam kegiatan kepanitiaan yaitu SCHEMATICS 2017 dan SCHEMATICS 2018 divisi Perlengkapan dan Transportasi (Perkaptrans). Penulis dapat dihubungi melalui email: chendrasena.oe@gmail.com.