Optimasi Jaringan Distribusi Listrik untuk Potensi EBT Mikrohidro di Desa Wonotoro Kecamatan Sambi dengan Algoritma Genetik

Sigit Khoirul Anam
Program Studi Kimia
Fakultas Matematika dan Ilmu Pengetahuan Alam
Universitas Negeri Yogyakarta
Sigitkhoirul.2020@student.uny.ac.id

Abstrak

Adanya mata air 'Sendang Siraman' di Desa Wonotoro memungkinkan untuk dijadikan Energi Baru Terbatukan (EBT), salah satunya PLTMH yang dapat membantu memenuhi kecukupan listrik khususnya di Desa Wonotoro, salah satu masalah yang mungkin dihadapi adalah mahalnya kabel yang digunakan untuk distribusi listrik kerumah warga selain itu penggunaan kabel yang terlalu panjang juga akan mengakibatkan rugi tegangan. Karya ilmiah ini akan mencoba menyelesaikan permasalahan tersebut menggunakan Algortima Genetik untuk mendapatkan penyaluran kabel yang optimal dengan mengurutkan node-node terdekat.

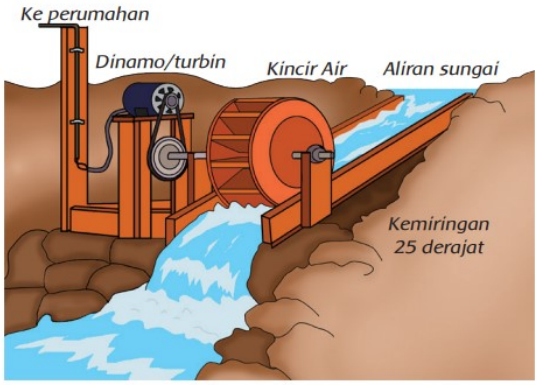
Data yang diperoleh adalah dari google map yang diolah dengan bantuan Geogebra untuk mendapatkan letak koordinat node sebagai letak rumah warga. Hasil pemrosesan algoritma menunjukkan bahwa kabel paling minimal bisa didapatkan dengan Algoritma Genetik 36000 generasi.

Kata kunci: Desa Wonotoro, EBT, Distribusi Jaringan, Algoritma Genetik.

A. Pendahuluan

Wonotoro merupakan suatu dukuh di Desa Catur, Kecamatan Sambi, Kabupaten Boyolali. Sebuah dukuh dengan jumlah penduduk lebih dari 100 orang, di daerah tersebut terdapat sebuah sendang yang dinamai dengan Sendang Siraman atau sebuah mata air. Mata air merupakan pemunculan air tanah ke permukaan tanah. Pemanfaatan mata air sangat beragam, antara lain penggunaan untuk keperluan air minum, irigasi, perikanan, untuk obyek wisata (Darmanto et al., n.d.) Mata air kebanyakan hanya mengalir begitu saja, belum sepenuhnya dimanfaatkan, atau kalau dimanfaatkan hanya sebatas untuk keperluan irigasi atau perikanan (Darmanto et al., n.d.). Sampai saat ini pemerintah setempat masih mengupayakan pembangunan sendang tersebut guna memberikan manfaat sebesar-besarnya bagi masyarakat sekitar. salah satu solusi yang dapat ditawarkan adalah sebagai sumber EBT Tenaga Mikrohidro (PLTMH) guna membantu mencukupi kebutuhan energi listrik di Desa Wonotoro.

Energi Baru Terbarukan (EBT) merupakan alternatif terbaik untuk mencukupi kebutuhan listrik yang ramah lingkungan disamping energi fosil yang semakin menispis (Azhar & Adam Satriawan, 2018). PLTMH sebagai pembangkit listrik pedesaan yang ramah lingkungan karena tidak menggunakan bahan bakar minyak (Rohermanto, 2007). Pembangkit Listrik Tenaga Mikrohidro (PLTMH) adalah teknologi untuk memanfaatkan debit air yang ada di sekitar kita untuk diubah menjadi energi listrik. Caranya adalah dengan memanfaatkan debit air untuk menggerakkan turbin yang akan menghasilkan energi mekanik.



Ilustrasi Pembangkit LIstrik Tenaga Mikrohidro (PLTMH)

Jaringan distribusi merupakan salah satu sistem dalam sistem tenaga listrik yang mempunyai peran penting karena berhubungan langsung dengan pemakai energi listrik (Fayyadl et al., n.d.). Jaringan distribusi sekunder digunakan untuk menyalurkan tenaga listrik dari gardu distribusi ke beban-

beban yang ada di konsumen. Distribusi energi listrik dari pusat pembangkit listrik (power plant) ke jaringan beban yang letaknya berjauhan selalu mengalami terjadiya rugi-rugi (losses), salah satunya adalah rugi tegangan. Rugi tegangan akan menyebabkan terjadinya jatuh tegangan (drop voltage) yang cukup besar, yang mengakibatkan rendahnya tegangan terima (Septi Yansuri & Ali Akbar, n.d.) oleh sebab itu salah satu cara untuk mengatasi permasalah tersebut adalah dengan optimasi minimal panjangnya penggunaan kabel listrik dari sumber listrik ke rumah-rumah(konsumen). Salah satu metode untuk menyelesaikan kasus optimasi yaitu algoritma Genetika (Azis et al., 2016).

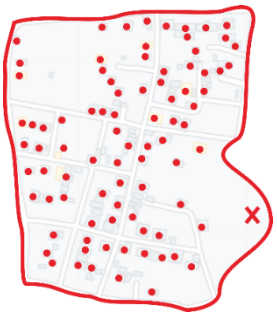
Algoritma genetika adalah algoritma pencarian heuristik yang didasarkan atas mekanisme seleksi alami dan genetika alami. Konsep dasar yang mengilhami timbulnya algoritma genetika adalah teori evolusi alam yang dikemukakan oleh Charles Darwin. Dalam teori tesebut dijelaskan bahwa pada proses evolusi alami, setiap individu harus melakukan adaptasi terhadap lingkungan disekitarnya agar dapat bertahan hidup (Kurnia et al., n.d.). Algoritma genetika mungkin tidak selalu mencapai hasil yang terbaik, tetapi seringkali memecahkan masalah dengan cukup baik. Algoritma genetika merepresentasikan suatu solusi permasalahan sebagai kromosom. Terdapat beberapa aspek penting dalam algoritma genetika antara lain defenisi fungsi fitness, defenisi dan implementasi representasi genetika, defenisi dan implementasi operasi genetika (Suprayogi & Mahmudy, n.d.).

D.E. Golberg dalam bukunya menggunakan metode Genetic Algorithm dalam menyelesaikan permasalahan optimasi di Industri. Metode GA kemudian berkembang penggunaanya ke berbagai bidang ilmu, salah satunya diaplikasikan dalam optimasi disain jaringan distribusi (Budiastra et al., 2006). Algoritma genetik pada kasus ini hampir mirip dengan kasus *Traveling Salesman Problem* (TSP).

B. Metode

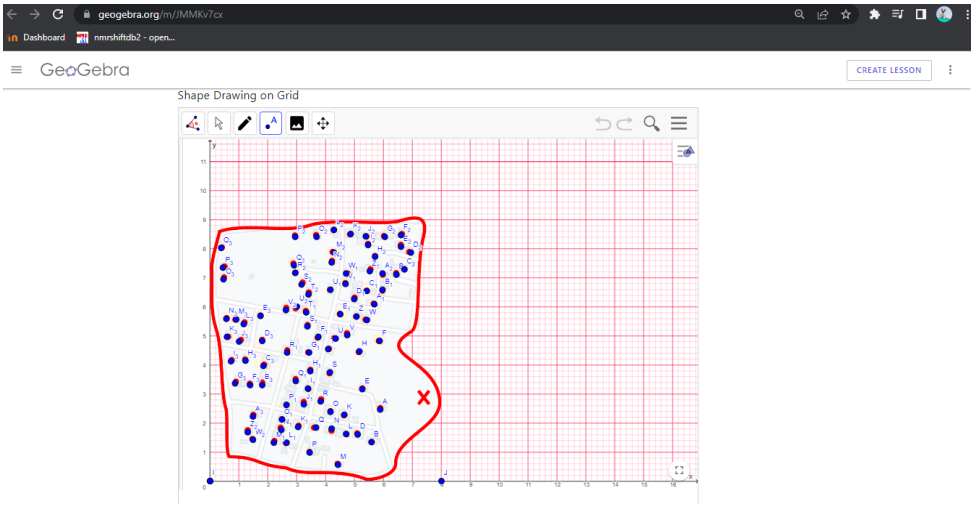
1. Lokasi

   Representasi denah atau lokasi diambil dari pencitraan Google Map untuk Dukuh Wonotoro, Desa Catur, Kecamatan Sambi, Kabupaten Boyolali. Sendang siraman berada di sebelah sisi kanan dari Dukuh Wonotoro. Dibawah ini adalah gambar dari denah rumah warga yang ditandai dengan node warna merah dan sendang siraman ditandai dengan tanda silang sebagai potensi EBT Mikrohidro, adapun garis merah adalah sebuah garis yang membatasi luas dari dukuh wonotoro sendiri



2. Penentuan Koordinat

Dari data yang ada terhitung jumlah rumah aktif adalah 86 rumah, sehingga ada 86 node yang perlu dihubungkan. Node-node yang terbentuk kemudian ditransformasikan kedalam bentuk koordinat kartesius dengan bantuan Geogebra yang dapat diakses secara online

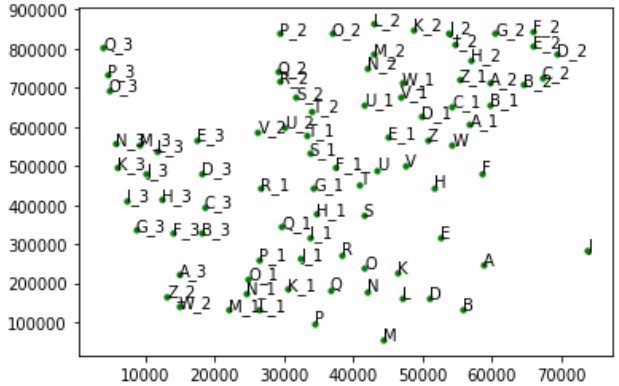


koordinat y
73825 , 58825 , 55825 , 51025 , 52625 , 58625 , 51625 , 46425 , 47025, 44225, 42025 , 41625, 34425 , 36625 , 38225 , 41425, 40825, 43425, 47425, 54225, 50625, 56825, 59625, 54225, 49825, 45025, 37425, 34225, 34625, 33825, 32425, 30425, 26425, 22025, 24625, 24825, 26425, 29625, 26625, 33625, 33225, 41625, 46825, 47025, 55225, 59625, 64425, 67225, 69425, 66025, 66025, 60425, 57025, 54625, 53825, 48625, 42825, 42825, 42025, 36825, 29425, 29025, 29425, 31625, 34025, 30000, 26225, 14825, 13025, 14825, 18025, 18425, 18025, 17425, 13825, 8625 , 12225, 7225, 10025, 5825 , 11625, 9025, 5625 , 4625, 4425 , 3825

koordinat y
284125,246125 , 134125, 160125, 316125, 482125, 446125, 228125, 162125, 56125, 180125, 238125, 98125, 184125, 274125,374125,454125,490125,502125,554125,566125,608125,658125,654125,626125,574125,496125,442125,378125,318125,264125,186125,132125,132125,176125,212125,262125,344125,442125,532125,580125,658125,678125,714125,722125,714125,710125,728125,786125,806125,846125,840125,772125,812125,842125,850125,864125,786125,752125,840125,840125,742125,716125,676125,642125,600000,588125,142125,168125, 222125,330125,396125,482125,568125,330125,336125,414125,412125,480125,496125,540125,556125,558125,694125,734125,802125])

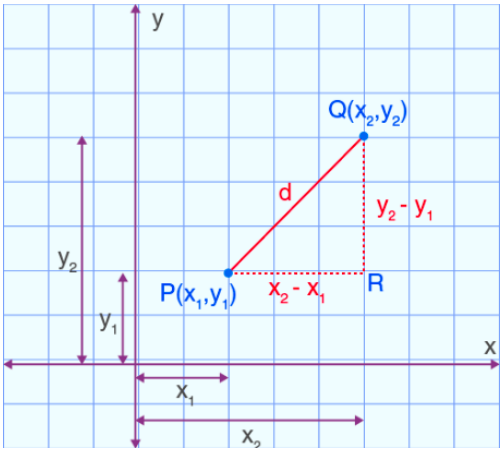
Label secara berurutan
["J","A","B","D","E","F","H","K","L","M","N","O","P","Q","R","S","T","U","V","W","Z","A_1","B_1","C_1","D_1","E_1","F_1","G_1","H_1","I_1","J_1","K_1","L_1","M_1","N_1","O_1","P_1","Q_1","R_1","S_1","T_1","U_1","V_1","W_1","Z_1","A_2","B_2","C_2","D_2","E_2","F_2","G_2","H_2","I_2","J_2","K_2","L_2","M_2","N_2","O_2","P_2","Q_2","R_2","S_2","T_2","U_2","V_2","W_2","Z_2","A_3","B_3","C_3","D_3","E_3","F_3","G_3","H_3","I_3","J_3","K_3","L_3","M_3","N_3","O_3","P_3","Q_3"]

Sehingga apabila diplot akan didapatkan model gambar dibawah ini, dengan node J sebagai lokasi Sendang Siraman.



3. Perhitungan jarak setiap node

Perhitungan fitness diperlukan untuk kebutuhan pada Algoritma Genetik guna mengetahui solusi terbaik. Perhitungan jarak antar node dilakukan dengan Formula *Euclide Distance* dan hasilnya disimpan kedalam sebuah matrix. Geometri Jarak Euclidean adalah studi tentang geometri Euclidean berdasarkan konsep jarak. Jarak Euclidian berguna dalam beberapa aplikasi di mana data input terdiri dari kumpulan jarak, dan outputnya adalah kumpulan titik dalam ruang Euclidean yang memberikan nilai jarak yang sesungguhnya (Liberti et al., 2014).



$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

dibawah ini menampilkan sebagian hasil perhitungan dari Euclide Distance untuk setiap node yang disimpan dalam sebuah matrix

```
[     0,  40853, 151076, ..., 415798, 455320, 522708]
[ 40853,      0, 112040, ..., 451266, 491022, 558713]
[151076, 112040,      0, ..., 562335, 602197, 670020]
...,
[415798, 451266, 562335, ...,      0,  40000, 108002]
[455320, 491022, 602197, ...,  40000,      0,  68002]
[522708, 558713, 670020, ..., 108002,  68002,      0]
```

4. Prosesing Algoritma Genetik

Algoritma Genetik yang digunakan mirip dengan kasus Traveling Salesman Trouble (TSP) yaitu pada proses crossover dan mutasinya yang tidak boleh ada nilai atau node/kota yang sama pada 1 kromoson.

a. Inisiasi

Jumlah Gen atau jumlah node adalah 86 (n), jumlah Kromoson adalah 100 (m) dan jumlah Generasi adalah 1000 (N) yang berarti bahwa 1 populasi berisi 100 kromoson (individu). Setiap selesai menjalankan 1000 generasi, generasi terakhir akan disimpan sebagai .CSV sehingga memungkinkan pelatihan berkelanjutan mengingat resource yang digunakan terbatas.

b. Pembangkitan populasi awal

Pembangkitan populasi awal dengan mengenerate cromoson yang diisi gen secara acak secara permutasi.

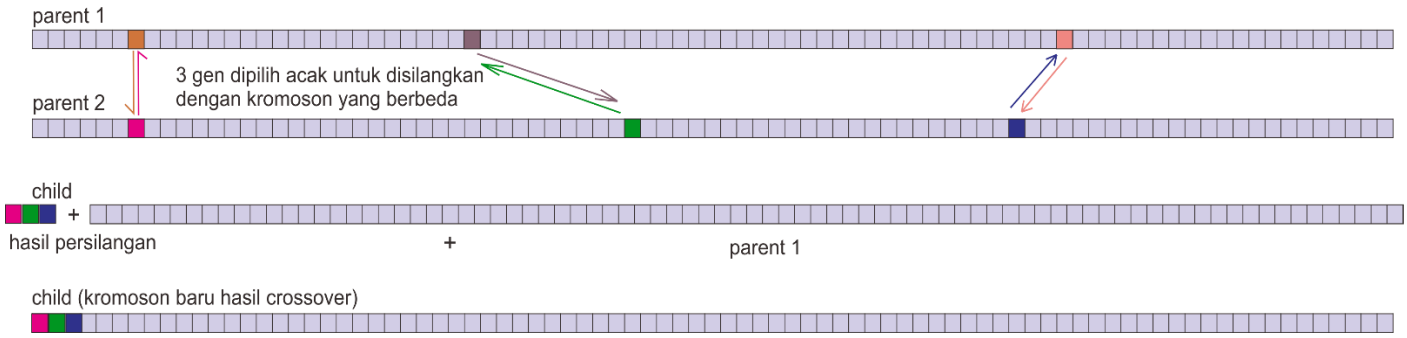
c. Fitness

Setiap populasi yang terbentuk akan dihitung nilai fitness untuk setiap kromoson dalam populasi. Perhitungan fitness yaitu dengan menjumlahkan semua nilai Euclide Distance dari node-node yang disusun sebagai solusi.

d. Crossover

Crossover terjadi setiap kali dua individu digabungkan untuk membuat individu baru dan akhirnya mereka disalin ke dalam populasi baru. Dua populasi yang dipilih didasarkan pada fungsi acak dan populasi ini bekerja sebagai induk. Dengan demikian orang tua menciptakan anaknya yang lebih baik dari orang tua itu sendiri dalam bentuk jarak dan akhirnya memproses populasi ini pada langkah selanjutnya (Sarkar, n.d.).

Dalam kasus ini child sebagai hasil offspring dari 2 parent. Child adalah hasil penggabungan hasil silang + parent 1 (ketika ada yang ganda nilai gen-nya maka dihilangkan kemudian ditambahi dengan apa yang tidak ada di parent 2), berikut adalah ilustrasinya

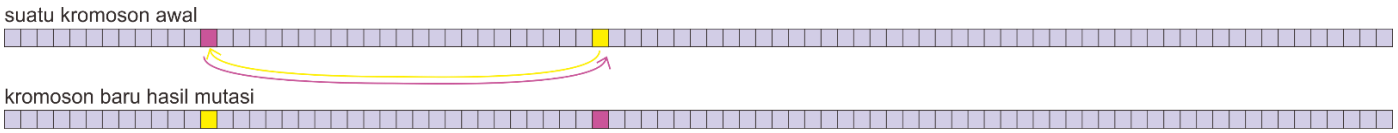


Gen maupun kromoson dipilih secara acak. kromoson dipilih acak sebanyak 2 kromoson, sedangkan gen dipilih sebanyak 3 gen yang akan disilangkan dari 2 kromoson terpilih (parent 1 & parent 2) hal tersebut dilakukan untuk setiap kromoson sehingga akan menghasilkan 1

populasi baru (100 kromoson atau individu). Crossover pada algoritma genetik diperlukan agar solusi yang diberikan tidak terjebak pada *local minimum* saja.

e. Mutasi

Mutasi merupakan bagian dari GA yang berkaitan dengan "eksplorasi" ruang pencarian. Telah diamati bahwa mutasi sangat penting untuk konvergensi GA sementara crossover tidak. Tetapi peran mutasi sering diremehkan di bidang Komputasi Evolusi (de Falco et al., 2002). Berikut adalah ilustrasi aplikasi mutasi untuk kasus ini



Mutasi dilakukan dengan menggantikan gen dengan gen lain dalam 1 jenis kromoson.

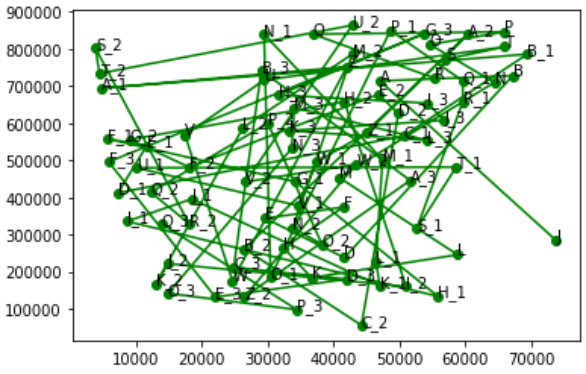
f. Penggabungan Populasi

Didapatkan 3 populasi yaitu populasi dari hasil acak (100 kromoson), populasi dari hasil crossover (100 kromoson), dan populais dari hasil mutasi (100 kromoson). Ketika populasi tersebut digabungkan sehingga jumlahnya adalah 300 kromoson (100 + 100 + 100). Dari 300 tersebut akan diambil 100 kromoson yang paling baik dengan memperhatikan nilai fitness yang paling kecil dengan metode sort, serta menghilangkan kromoson yang memiliki solusi yang sama dengan kromoson lain.

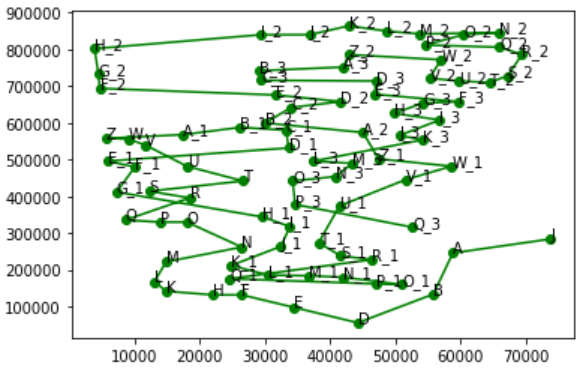
g. *Running* dan *plotting*

Running Algoritma ini sebanyak 36000+ generasi untuk mendapatkan solusi yang optimal, program ini dijalankan membutuhkan waktu 1-2 hari sehingga perlu dipotong-potong dengan cara menyimpan solusi pada generasi terakhir kedalam file .CSV, sehingga ketika ingin melanjutkan pada generasi selanjutnya adalah dengan menggunakan acuan file CSV tersebut yang telah disimpan.

C. Hasil dan Diskusi

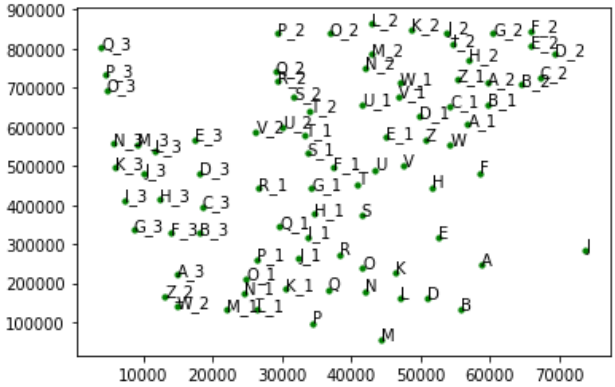


Plotting solusi terbaik pada generasi 1, fitness = 22.659.392



Plotting solusi terbaik pada generasi 36000+, fitness = 2.837.768

Didapatkan solusi terbaik dengan memasang kabel sesuai urutan rumah yaitu:



['J', 'A', 'B', 'M', 'P', 'L_1', 'M_1', 'W_2', 'Z_2', 'A_3', 'P_1', 'B_3', 'F_3', 'G_3', 'C_3', 'H_3', 'R_1', 'D_3', 'L_3', 'M_3', 'N_3', 'E_3', 'V_2', 'T_1', 'S_1', 'K_3', 'J_3', 'I_3', 'Q_1', 'I_1', 'J_1', 'O_1', 'K_1', 'Q', 'N', 'D', 'L', 'N_1', 'K', 'O', 'R', 'S', 'H', 'F', 'V', 'E_1', 'U_2', 'T_2', 'U_1', 'S_2', 'O_3', 'P_3', 'Q_3', 'P_2', 'O_2', 'L_2', 'K_2', 'J_2', 'F_2', 'G_2', 'I_2', 'E_2', 'D_2', 'C_2', 'B_2', 'A_2', 'Z_1', 'H_2', 'M_2', 'N_2', 'Q_2', 'R_2', 'W_1', 'V_1', 'B_1', 'C_1', 'D_1', 'A_1', 'Z', 'W', 'F_1', 'U', 'T', 'G_1', 'H_1', 'E']

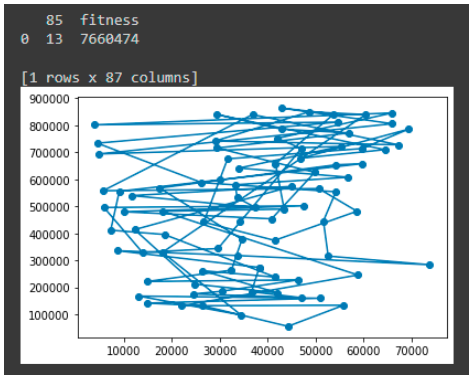
Banyaknya kromoson pada Algoritma Genetik membuat algoritma membutuhkan waktu yang sangat lama untuk mendapatkan solusi terbaik, disamping itu penerapan perhitungan dan operator-operator fitness sulit diaplikasikan pada algoritma ini.

Solusi dengan fitness 2.837.768 didapatkan setelah 3600+ generasi, proses algoritma dihentikan karena dalam 3000 generasi terakhir tidak terjadi

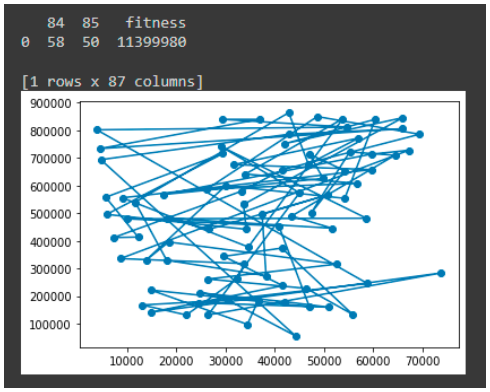
perubahan nilai fitness. Dengan demikian dari 3600+ generasi terjadi optimasi fitness dari 22.659.392 ke 2.837.768 (berkurang 19.821.624)

Pada sesi crossover, hanya dipilih 3 gen secara acak dari 86 gen, karena menurut pengalaman ini lebih memberikan dampak pada penurunan fitness daripada 15 gen (pada 200 generasi).

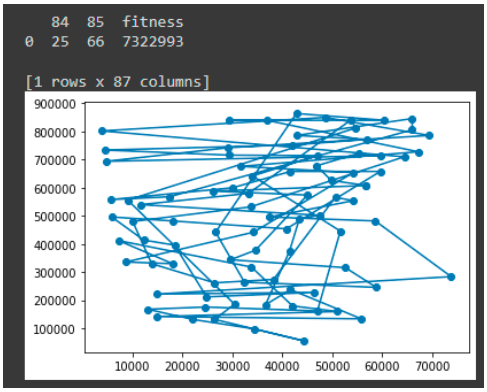
Ketika Algoritma Genetik hanya menggunakan mutasi saja, itu bisa bisa disebut dengan *Evolutioin Algorithm*, untuk pengujian dilakukan dengan 200 generasi, percobaan pertama tidak menggunakan crossover hasilnya adalah didapatkan fitness 7.660.474 direpresentasikan pada gambar dibawah ini



Ketika hanya menggunakan crossover tanpa mutasi didapatkan fitness yang lebih jelek daripada hanya menggunaakn mutasi saja yaitu 11.399.980



Sedangkan ketika digabungkan crossover dengan mutasi memberikan nilai fitness paling baik yaitu 7.322.993



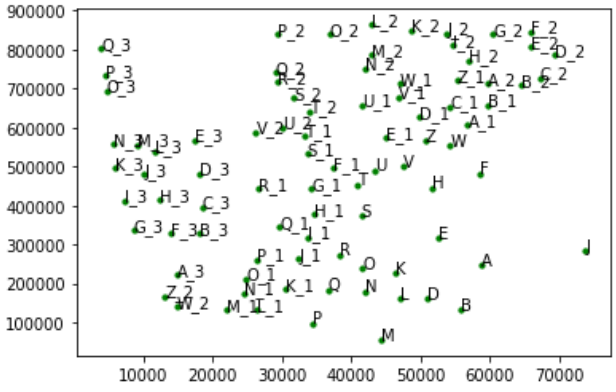
Eksekusi algoritma dilakukan di Google Colab untuk medapatkan resource yang cukup mumpuni, hanya saja maksimal penggunaan di Google Colab adalah 12 jam, setelah itu runtime akan terputus dan mengulang data dari awal sehingga diperlukan export generasi terakhir agar dapat dilakukan pelatihan secara kontinu.

Sebanyak 86 node diolah dengan Geogebra dihasilkan koordinat-koordinat kartesian dengan label-labelnya, Geogebra dapat diakses secara online, pertama kali adalah memasukkan gambar hasil pencitraan Google Map ke Geogebra, kemudian diberikan tanda pada setiap node serta labelnya sehingga akan didapatkan file asymetric.txt dari Geogebra yang memuat letak node dan labelnya, koordinat letak node dan label akan diekstraksi dengan pemograman terpisah.

D. Kesimpulan

Optimasi jaringan distribusi listrik untuk potensi EBT mikrohidro 86 rumah di Desa Wonotoro Kecamatan Sambi dapat diselesaikan dengan Algoritma Genetik dengan dengan 3600 generasi.

Urutan rumah atau node adalah ['J', 'A', 'B', 'M', 'P', 'L_1', 'M_1', 'W_2', 'Z_2', 'A_3', 'P_1', 'B_3', 'F_3', 'G_3', 'C_3', 'H_3', 'R_1', 'D_3', 'L_3', 'M_3', 'N_3', 'E_3', 'V_2', 'T_1', 'S_1', 'K_3', 'J_3', 'I_3', 'Q_1', 'I_1', 'J_1', 'O_1', 'K_1', 'Q', 'N', 'D', 'L', 'N_1', 'K', 'O', 'R', 'S', 'H', 'F', 'V', 'E_1', 'U_2', 'T_2', 'U_1', 'S_2', 'O_3', 'P_3', 'Q_3', 'P_2', 'O_2', 'L_2', 'K_2', 'J_2', 'F_2', 'G_2', 'I_2', 'E_2', 'D_2', 'C_2', 'B_2', 'A_2', 'Z_1', 'H_2', 'M_2', 'N_2', 'Q_2', 'R_2', 'W_1', 'V_1', 'B_1', 'C_1', 'D_1', 'A_1', 'Z', 'W', 'F_1', 'U', 'T', 'G_1', 'H_1', 'E']



E. Referensi

Azhar, M., & Adam Satriawan, D. (2018). Implementasi Kebijakan Energi Baru dan Energi Terbarukan Dalam Rangka Ketahanan Energi Nasional. In *Online Administrative Law & Governance Journal* (Vol. 1).

Azis, A., Prihandono, B., & Intisari, I. (2016). Algoritma Genetika Pada Pemrograman Linear Dan Nonlinear. In *Buletin Ilmiah Mat. Stat. dan Terapannya (Bimaster)* (Vol. 5, Issue 03).

Budiastra, N., Penangsang, O., Mauridhi, D., & Purnomo, H. (2006). Optimasi Jaringan Distribusi Sekunder Untuk Mengurangi Rugi Daya Menggunakan Algoritma Genetika. *Seminar Nasional Aplikasi Teknologi Informasi*.

Darmanto, D., Widyastuti, M., & Sri Lestari, dan. (n.d.). Pengelolaan Mata Air Untuk Penyediaan Air Rumahtangga Berkelanjutan Di Lereng Selatan Gunungapi Merapi (Springs Management for Sustainability Domestic Water Supply in the Southwest of Merapi Volcano Slope). In *Maret* (Vol. 23, Issue 1).

de Falco, I., della Cioppa, A., & Tarantino, E. (2002). Mutation-based genetic algorithm: Performance evaluation. *Applied Soft Computing*, *1*(4), 285–299. https://doi.org/10.1016/S1568-4946(02)00021-2

Fayyadl, M., Sukmadi, I. T., & Winardi, I. B. (n.d.). *Rekonfigurasi Jaringan Distribusi Daya Listrik Dengan Metode Algoritma Genetika*.

Kurnia, N., Wayan, M., Mahmudy, F., & Matematika, J. (n.d.). *Optimasi Penjadwalan Ujian Menggunakan Algoritma Genetika* (Vol. 2, Issue 2).

Liberti, L., Lavor, C., Maculan, N., & Mucherino, A. (2014). Euclidean distance geometry and applications. *SIAM Review*, *56*(1), 3–69. https://doi.org/10.1137/120875909

Rohermanto, A. (2007). *Pembangkit Listrik Tenaga Mikrohidro (PLTMH) AGUS ROHERMANTO* (Vol. 4, Issue 1).

Sarkar, S. (n.d.). *Implementation of Parallel Genetic Algorithm Word Count problem using MapReduce in Hadoop View project Implementation of Parallel Genetic Algorithm View project Implementation of Parallel Genetic Algorithm*. https://www.researchgate.net/publication/340579712

Septi Yansuri, D., & Ali Akbar, M. (n.d.). *Evaluasi Jenis Kabel Dari Tembaga Ke Alumunium Untuk Distribusi Power Supply 20 Kv Coal Conveyor*.

Suprayogi, D. A., & Mahmudy, W. F. (n.d.). *Penerapan Algoritma Genetika Traveling Salesman Problem with Time Window: Studi Kasus Rute Antar Jemput Laundry 121*.

SCRIPT GENETIC ALGORITHM

Bahasa pemograman : Python 3.9.1

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

x = np.array([73825 , 58825  , 55825 , 51025 , 52625 , 58625 , 51625
, 46425 , 47025, 44225,  42025 , 41625, 34425 , 36625 , 38225,
41425, 40825, 43425, 47425, 54225, 50625, 56825, 59625, 54225,
49825, 45025, 37425, 34225, 34625, 33825, 32425, 30425, 26425,
22025, 24625, 24825, 26425, 29625, 26625, 33625, 33225, 41625,
46825, 47025, 55225, 59625, 64425, 67225, 69425, 66025, 66025,
60425, 57025, 54625, 53825, 48625, 42825, 42825, 42025, 36825,
29425, 29025, 29425, 31625, 34025, 30000, 26225, 14825, 13025,
14825, 18025, 18425, 18025, 17425, 13825, 8625 , 12225, 7225 ,
10025, 5825 , 11625, 9025, 5625  , 4625, 4425  , 3825])
y = np.array([284125,246125, 134125, 160125, 316125, 482125, 446125,
228125, 162125, 56125, 180125, 238125, 98125, 184125,
274125,374125,454125,490125,502125,554125,566125,608125,658125,65412
5,626125,574125,496125,442125,378125,318125,264125,186125,132125,132
125,176125,212125,262125,344125,442125,532125,580125,658125,678125,7
14125,722125,714125,710125,728125,786125,806125,846125,840125,772125
,812125,842125,850125,864125,786125,752125,840125,840125,742125,7161
25,676125,642125,600000,588125,142125,168125,222125,330125,396125,48
2125,568125,330125,336125,414125,412125,480125,496125,540125,556125,
558125,694125,734125,802125])
anotasi =
["J","A","B","D","E","F","H","K","L","M","N","O","P","Q","R","S","T"
,"U","V","W","Z","A_1","B_1","C_1","D_1","E_1","F_1","G_1","H_1","I_
1","J_1","K_1","L_1","M_1","N_1","O_1","P_1","Q_1","R_1","S_1","T_1"
,"U_1","V_1","W_1","Z_1","A_2","B_2","C_2","D_2","E_2","F_2","G_2","
H_2","I_2","J_2","K_2","L_2","M_2","N_2","O_2","P_2","Q_2","R_2","S_
2","T_2","U_2","V_2","W_2","Z_2","A_3","B_3","C_3","D_3","E_3","F_3"
,"G_3","H_3","I_3","J_3","K_3","L_3","M_3","N_3","O_3","P_3","Q_3"]

for i, label in enumerate(anotasi): #plotting berdasarkan koordinat
x, y serta labelnya
    plt.annotate(label, (x[i], y[i]))
plt.scatter(x, y, s=10, color='green')
plt.show()

# number of cities / points / node
```

```
m = len(x)
# number of chromosomes in population
n = 100
# maximum generation
N = 1000

# Jarak antar nod yang disimpan dalam matrix
d = np.zeros((m, m), dtype=int)
for i in range(m):
    for j in range(m):
        d[i, j] = np.sqrt((x[i] - x[j])**2 + (y[i] - y[j])**2)
#formula Euclide Distance
d

def createPopulation():
    pop = np.zeros((n, m), dtype=int)
    #hidupkan nomor 1 jika memulai dari awal | hidupkan nomor 2 jika
memulai dari weight yang sudah ada
    #[1]
    # for i in range(n):
    #     pop[i] = np.random.permutation(m)
    #     pop[0] = []
    # pop = pd.DataFrame(pop)
    #[2]
    oo = pd.read_csv('/content/weight.csv') #membaca file weight
    for i in range(n):
        pop[i] = oo.loc[i].values.tolist() #ini dalam bentuk list
yang akan menggantikan zeros dengan nilai yang ada di weight
    pop = pd.DataFrame(pop) #kalau oo[1] -> itu kolom | kalau
oo.loc[1] -> itu baris | .values.tolist mengubah ke list
    return pop

def fitness(pop):
    fitness = np.zeros(n, dtype=int) #membuat kolom yang berisi
nilai nol
#mengisi kolom yang tadinya nol dengan jarak2 yang telah dihitung
pada matrix diatas
    for k in range(n):
        a = pop.loc[k]
        b = 0
        for i in range(0, m-1):
            b += d[a[i], a[i+1]]
        b += d[a[m-1], a[0]]
        fitness[k] = b
```

```
        pop['fitness'] = fitness #memasukkan hasil perhitungan
fitness kedalam kolom baru bernama fitness
    #untuk setiap kromoson
    return pop


def crossover(pop):
    popc = pop.copy()
    for o in range(n):
        kposition = np.random.permutation(n)
        acakkrom1 = kposition[0] #kromoson1 dengan kromoson2
sehingga akan mutlak berbeda
        acakkrom2 = kposition[1]
        gposition = np.random.permutation(m)
        acakgen1 = gposition[0]
        acakgen2 = gposition[1]
        acakgen3 = gposition[2]

        isiindex1 = popc.at[acakkrom1,acakgen1] #memilih 1 jenis
kromoson, dipilih acak genya, dijadikan sample untuk persilangan
        isiindex2 = popc.at[acakkrom1,acakgen2]
        isiindex3 = popc.at[acakkrom1,acakgen3]

        parent1 = popc.loc[acakkrom1]
        parent2 = popc.loc[acakkrom2]
        childP1 = [isiindex1,isiindex2,isiindex3] # ini menunjukkan
hasil dari index

        for i in range(n,m): #kromoson/baris(n), gen/column(m)
            childP1.append(parent1[i])
        childP2 = [item for item in parent2 if item not in childP1]
        res = []
        for i in childP1: #apakah ada yang double, disek disini
            if i not in res:
                res.append(i)
        child = res + childP2
        popc.loc[o] = child
    return popc

def mutation(pop):
    popm = pop.copy()
    for i in range(n):
        position = np.random.permutation(m)
        a = position[0]
        b = position[1]
```

```
            temp = popm.loc[i][a]
            popm.loc[i][a] = popm.loc[i][b] #meng exchange-kan antar gen
dalam 2 jenis kromoson,
            popm.loc[i][b] = temp #ini dilakukan untuk setiap kromoson,
begitu juga crossover tadi
    return popm

def combinePopulation(pop, popm, popc):
    popAll = pop.copy() #menggabungkan populasi yang dibuat dari
mutasi crosover dan populasi awal
    popAll = popAll.append(popm)
    popAll = popAll.append(popc)
    popAll = popAll.drop_duplicates() #sehingga ada 300 kromoson,
yang memiliki nilai sama persis akan dihapus(drop)
    popAll.index = range(len(popAll))
    return popAll

def sort(popAll):
    popAll = popAll.sort_values(by=['fitness']) #mengurutkan
berdasarkan fitness, kromoson yang memiliki fitness
    #rendah akan semakin diatas
    popAll.index = range(len(popAll))
    return popAll

def elimination(popAll):
    pop = popAll.head(n) #mengambil kromoson paling atas guna
keperluan plotting
    return pop

def plotSolution(pop): #plotting solusi
    solution = pop.loc[0]
    solution = solution.to_numpy()
    a = np.zeros(m, dtype=int)
    b = np.zeros(m, dtype=int)
    for i in range(m):
        a[i] = x[solution[i]]
        b[i] = y[solution[i]]
    for i, label in enumerate(anotasi):
      plt.annotate(label, (a[i], b[i]))
    plt.plot(a, b, color='green', marker = 'o')
    plt.show()

pop = createPopulation() #eksekusi dengan memanggil fungsi2 yang
telah dibuat
pop = fitness(pop)
```

```
print('Solusi pada populasi awal')
print(pop.head(1))
plotSolution(pop)
for i in range(1, N+1):
    popc = crossover(pop)
    popc = fitness(popc)
    popm = mutation(pop)
    popm = fitness(popm)
    popAll = combinePopulation(pop, popm, popc)
    popAll = sort(popAll)
    pop = elimination(popAll)
    print()
    print('Solusi terbaik pada populasi generasi ke-'+ str(i))
    print(pop.head(1))
    plotSolution(pop)
print()
print('Solusi terbaik pada populasi akhir')
print(pop.head(1))
plotSolution(pop)
```