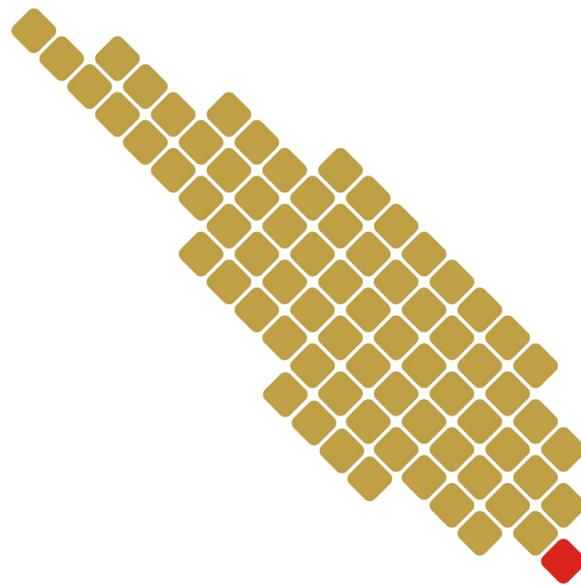


[Implementasi Cuucko Search Terhadap Jarak Terbaik  
Kota Jakarta Menuju Malang Dengan Metode Travelling  
Salesman Problem]

Laporan akhir *swarm intellegence*



**ITERA**

**Kelompok {4}**

**Ketua Kelompok : {120450032} - {Vicky aditya}**  
**Anggota 1 : {120450090} - {Nabil Azizi}**  
**Anggota 2 : {120450050} - {M Dafha Syahrizal}**  
**Anggota 3 : {120450058} - {M Aditya Putra Suhendar}**

## INTRODUCTION

Seiring perkembangan zaman dan kemajuan teknologi sistem informasi membuat pelayanan jasa pengiriman barang yang dilakukan kantor pos kini lebih sistematis. Kantor pos sekarang tidak hanya mengantarkan surat tetapi juga mengantarkan barang paket dari pengirim yang ditujukan ke suatu kota tertentu. Dalam hal ini kurir harus mengirimkan banyak barang ke kota pelanggan dengan pertimbangan efisiensi waktu.

Sedangkan pada jaman sekarang, jumlah jalan yang sangat banyak terkadang menyulitkan petugas pos untuk dapat memilih jalur yang cepat ke kota tujuan pelanggan. Biasanya jalur yang ditempuh hanyalah jalur yang dihafal dan sering dilalui serta dianggap terpendek, padahal belum tentu jalur tersebut lebih optimal.

Oleh sebab itu diperlukan sebuah sistem yang memanfaatkan teknologi informasi guna melakukan proses pencarian jalur optimum secara otomatis. Travelling Salesman Problem (TSP) merupakan masalah seorang salesman yang ingin mengunjungi beberapa tempat dimana kembali lagi ke tempat asalnya tepat satu kali sehingga diperoleh jarak terpendek.

## PROBLEM STATEMENT

Dengan adanya probabilitas tiap kota dan jumlah kota yang tidak sedikit maka akan ada banyak kombinasi rute yang mungkin muncul dan harus dihitung. Bila menggunakan perhitungan eksak maka memerlukan waktu yang lama, sehingga untuk mempersingkat digunakanlah pendekatan heuristic. Pendekatan heuristic yang dipakai dalam penelitian ini adalah Algoritma Cuckoo Search (CS). Algoritma ini terinspirasi dari cara burung cuckoo berkembang biak. Algoritma CSO (Cuckoo Search Optimization) adalah salah satu algoritma optimasi yang dapat digunakan untuk pengambilan keputusan. Algoritma ini meniru tingkah laku dari spesies cuckoo yaitu sebuah parasit yang meletakkan telurnya di sarang burung lain (yang tentu saja bukan spesies cuckoo). Jika induk burung menemukan telur yang bukan dari dirinya sendiri, maka induk burung tersebut akan membuang

telur parasit tersebut atau membangun sarang baru ditempat lain. Cuckoo yang berhasil tumbuh nantinya akan mencari sarang burung lain sebagai tempat peletakan telur. Proses tersebut berulang sampai semua cuckoo sudah berkumpul pada sebuah sarang burung.

Berdasarkan identifikasi tersebut, muncul masalah yang dirumuskan

1. Berapa banyak rute dari jakarta ke malang?
2. Bagaimana algoritma CS menyeleksi rute terbaik dari jakarta ke malang?
3. Apa rute terbaik yang disimpulkan berdasarkan hasil algoritma CS?

## DATA DESCRIPTION

Pada algoritma CS dan dengan metode TSP kita menggunakan data tentang data Jakarta dengan beberapa kota di pulau jawa, salah satu nya yaitu kota Malang.

Kami menggunaan data beberapa kota yang ada di pulau jawa, beserta jaraknya. dikarenakan kami ingin mencari tahu kota apa saja yang harus ditempuh untuk ke malang agar mendapatkan jarak terbaik atau jarak yang paling efisien.

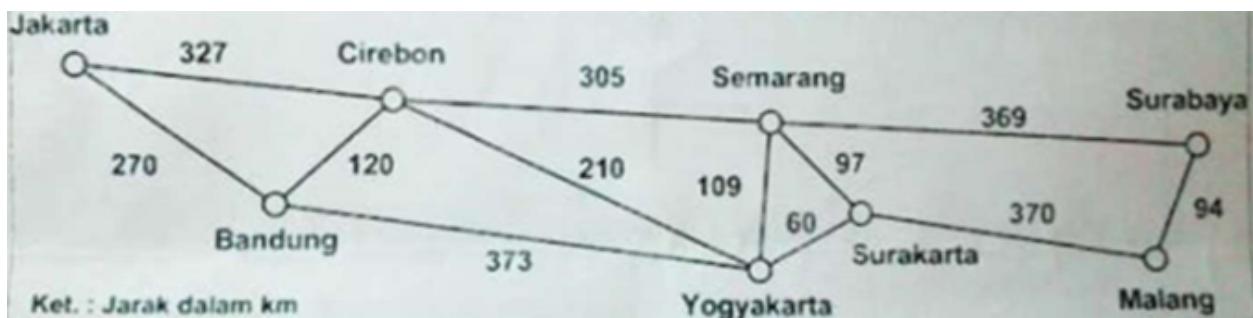
Kami tampilkan data sebagai berikut:

Tahun	Nama Provinsi	Nama Kota	Jarak
2015	Jawa Barat	Bandung	270
2015	Jawa Barat	Cirebon	327
2015	Yogyakarta	Yogyakarta	582
2015	Jawa Tengah	Semarang	482
2015	Jawa Tengah	Surakarta	657
2015	Jawa Timur	Surabaya	789
2015	Jawa Timur	Malang	964

- Matriks ketetanggaan

	JKT	BDG	Cirebon	Yogya	SMRG	SRKT	SRBY	MLG
Jakarta	0	1	1	0	0	0	0	0
Bandung	1	0	1	1	0	0	0	0
Cirebon	1	1	0	1	1	0	0	0
Yogyakarta	0	1	1	0	1	1	0	0
Semarang	0	0	1	1	0	1	1	0
Surakarta	0	0	0	1	1	0	0	1
Surabaya	0	0	0	0	1	1	0	1
Malang	0	0	0	0	0	1	1	0

- Graf matriks ketetanggaan



## METHOD

### 1. Metode algoritma Swarm Intelligence (SI) Cuckoo Search

Algoritma cuckoo search adalah salah satu dari Algoritma yang terinspirasi dari alam, yaitu terinspirasi dari sifat parasit beberapa spesies cuckoo yang meletakkan telurnya di sarang burung inang lainnya. Ada tiga pendekatan yang digunakan dalam Algoritma Cuckoo Search dirumuskan sebagai berikut :

- Setiap cuckoo meletakkan satu telur pada satu waktu, dan membuang telur dalam sarang yang dipilih secara acak.
- Sarang terbaik dengan telur yang berkualitas sebagai solusi yang akan digunakan untuk generasi selanjutnya.
- Jumlah sarang burung lain yang tersedia adalah tetap, dan burung pemilik sarang dapat menemukan telur cuckoo dengan probabilitas  $pa \in (0,1)$ . Dalam kasus ini, burung pemilik sarang dapat membuang telur cuckoo atau

meninggalkan sarang sehingga pemilik sarang dapat membangun sarang baru di lokasi yang baru.

Berikut ini adalah beberapa istilah yang digunakan dalam Cuckoo Search Algorithm (CSA) :

- (a) Telur Diasumsikan bahwa cuckoo meletakkan satu telur pada satu sarang, dan satu telur tersebut merupakan sebuah solusi yang direpresentasikan oleh satu individu (sarang) dalam populasi. Sebuah telur dapat menjadi salah satu calon solusi baru untuk sebuah tempat atau lokasi yang telah dimiliki oleh satu individu lain dalam populasi tersebut.
- (b) Sarang Sebuah sarang merupakan individu dari populasi yang memiliki jumlah tetap dan merepresentasikan besar populasi sehingga pada pergantian sarang akan melibatkan penggantinya dalam populasi dengan individu baru.
- (c) Fungsi Tujuan Setiap solusi dalam ruang pencarian dikaitkan dengan nilai nilai objektif numeric sehingga kualitas nilainya sebanding dengan nilai fungsi tujuan. Dalam Cuckoo Search Algorithm (CSA), sarang telur dengan kualitas yang lebih baik akan mengarah pada generasi baru yang berarti kualitas telur kukuk berhubungan langsung untuk kemampuannya memberikan cuckoo baru.
- (d) Ruang Pencarian Ruang pencarian merepresentasikan posisi dari sarang yang mungkin pada daerah feasible. Posisi sarang dapat diubah dengan memodifikasi nilai dari koordinatnya. Dapat dipastikan bahwa perpindahan sarang atau lokasi sarang tidak melebihi batasan yang sebenarnya.
- (e) Random Walks Pada dasarnya, burung mencari makanan secara acak di alam bebas. Cara paling efektif untuk seekor burung mencari makanannya adalah dengan menggunakan strategi Random Walks, karena langkah selanjutnya ditentukan berdasarkan lokasi saat ini dan peluang peralihan ke tempat selanjutnya

Secara umum, cara yang paling efektif untuk seekor burung mencari makanannya adalah dengan menggunakan strategi Random Walks, karena langkah selanjutnya ditentukan berdasarkan lokasi saat ini dan peluang

peralihan ke tempat selanjutnya. Cuckoo Search menggunakan dua random walks, diantaranya Levy Flights Random Walks (LFRW) dan Biased Random Walks (BRW).

Seperti yang kita ketahui Traveling Salesman Problem (TSP) adalah sebuah metode dengan permasalahan yaitu seorang salesman harus mengunjungi seluruh kota dimana tiap kota hanya akan dikunjungi sebanyak sekali, Tujuan dari permasalahan ini juga meminimumkan total jarak yang ditempuh salesman dengan mengatur urut-urutan kota yang harus dikunjungi. Hal tersebutlah yang membuat penggunaan algoritma Levy Flights Random Walk (LFRW) sangat cocok digunakan pada permasalahan ini.

Levy Flights Random Walk merupakan random walk dengan step length berdasarkan distribusi Levy. Levy Flights merupakan pencarian solusi baru yang berada disekitar solusi terbaik sementara dan dilakukan berdasarkan persamaan :  $x_i t+1 = x_i t + \alpha \cdot S \cdot (x_{best} t - x_i t) \cdot \kappa$  dengan  $x_i t$  merupakan sarang ke  $i$  pada iterasi ke  $t$ ,  $\alpha > 0$  merupakan parameter step length,  $S$  merupakan Levy flights dari algoritma Mantegna,  $\kappa$  merupakan bilangan real acak yang berdistribusi normal dengan rata-rata 0 dan simpangan baku 1 ,  $x_{best} t$  merupakan sarang terbaik pada iterasi ke  $t$ .

## 2. Pseudocode dari metode algoritma cuuckoo search

Berikut adalah Algoritma dasar dari Cuckoo Search :

---

**Cuckoo search algorithm**

---

Define Objective function  $f(x)$ ,  $x = (x_1, x_2, \dots, x_d)$   
Initial a population of  $n$  host nests  $x_i (i=1,2,\dots,d)$   
**while** ( $t < MaxGeneration$ ) or (stop criterion);  
Get a cuckoo (say  $i$ ) randomly  
    and generate a new solution by Lévy flights;  
    Evaluate its quality/fitness;  $F_i$   
Choose a nest among  $n$  (say  $j$ ) randomly;  
if ( $F_i > F_j$ ),  
    Replace  $j$  by the new solution;  
end  
Abandon a fraction ( $P_a$ ) of worse nests  
[and build new ones at new locations via Lévy flights];  
Keep the best solutions (or nests with quality solutions);  
Rank the solutions and find the current best;  
**end while**  
Post process results and visualization;  
**End**

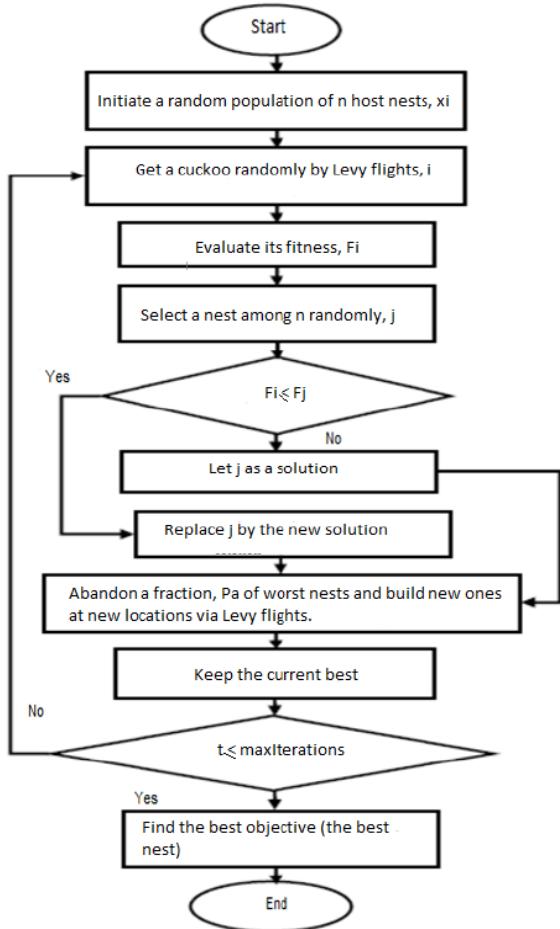
---

### 3. Modifikasi dari metode yang digunakan

Mengenai Modifikasi dari metode Cuucko Search, dilakukan pengolahan data sesuai dengan permasalahan yang ada. Dalam hal ini, dilakukan pembahasan dan analisis terhadap hasil pengolahan data, adapun metode yang digunakan untuk menentukan jarak atau rute terpendek yaitu:

- Cuckoo Search With Levy Flight Algorithm
- Hybrid Cuckoo Search with Levy Flight Algorithmdari metode yang

### FLOWCHART DARI METODE CUUCKO SEARCH



## RESULT AND DISCUSSION

## *Individual modelling*

```
▶ n_kota = len(df)
n_individu = 15
a = 60
b = 400

gen_individu = lambda n_individu, n_kota,a,b: np.random.uniform(a,b,(n_individu,n_kota))
cuckoos = gen_individu(n_individu,n_kota,a,b)
cuckoos

[+] array([[140.06377308, 332.39412091, 281.80066653, 330.08240162,
       255.58027115, 67.25294051, 219.54067258, 190.43107131],
      [291.46664968, 224.87416711, 101.2568801, 182.08165315,
       159.68333327, 316.69566016, 85.15239173, 77.05647636],
      [363.46520045, 307.56238048, 315.01192003, 380.12839228,
       124.73677348, 188.12608629, 116.79754729, 280.07296423],
      [238.87489108, 203.05140932, 295.81756829, 294.13358322,
       268.84839551, 235.03332314, 233.89630098, 149.85236118],
      [ 83.99275603, 383.46455287, 365.36201496, 187.40314486,
       120.8268142 , 353.14715275, 259.63980519, 115.42033233],
      [328.7417526 , 174.09045599, 358.6167034 , 188.95273224,
       223.00521234, 96.77490164, 366.80991582, 128.31401348],
      [390.08402808, 299.02849822, 174.79352139, 342.97388183,
       170.00386347, 373.92135393, 78.95697407, 148.53737496],
      [309.44886466, 350.75582624, 290.86515308, 315.15487376,
       225.91878262, 324.37748216, 139.21340998, 273.11426513],
      [382.31389268, 355.64521513, 170.41531515, 119.66813454,
       87.08110578, 252.90021703, 329.67454805, 188.51108707],
      [133.89756851, 97.5773294 , 283.91806665, 324.652677 ,
       143.40800977, 397.5851029 , 217.90253108, 155.06751107],
      [167.84604252, 214.97804479, 219.05303263, 151.80681274,
       128.5147443 , 204.34966392, 166.24470408, 287.34835313],
      [132.73719409, 87.93107416, 97.49783474, 372.15145079,
       83.0630078 , 327.73604183, 79.14853894, 393.04268377],
      [ 66.87319497, 109.90138846, 175.23199868, 146.55371034,
       72.44948813, 280.15409395, 271.18320786, 176.53804513],
      [357.13234875, 224.55913387, 250.09276862, 185.09898195,
       389.56900788, 271.56972036, 64.97679338, 327.17999549],
      [ 87.41538235, 306.67264643, 331.47504252, 390.22268588,
       385.20233679, 275.52105869, 381.68395724, 213.61939901]])
```

```
[+] def diskritisasi(cuckoos):
    return np.argsort(cuckoos)

[+] def calculate_fitness(cuckoos,df):
    d_cuckoos = diskritisasi(cuckoos)
    fitness = np.array( list(map(lambda x: calc_dist( x ,df.values), d_cuckoos )) )
    fitness = fitness.reshape( (-1,1) )
    return np.concatenate( ( cuckoos ,fitness ) ,axis=1)

[+] def sort_individu(cuckoos_with_f):
    return cuckoos_with_f[cuckoos_with_f[:,1].argsort()]

[+] cuckoos_w_f = sort_individu(calculate_fitness(cuckoos,df))

[+] def solusi(cuckoos_w_f):
    df_kota = pd.DataFrame(diskritisasi(cuckoos_w_f[:, :-1]))
    cols = [ 'Urutan ' + str(i+1) for i in range( df_kota.shape[1]) ]
    df_kota.columns = cols
    df_kota['Jarak'] = cuckoos_w_f[:, -1].reshape(-1,1)
    return df_kota

▶ solusi(cuckoos_w_f)
```

## Result and Analysis

Solusi terbaik, parameter terbaik, iterasi berhenti

```

n_iter = 100
generasi = 0
n_kota = len(df)
n_individu = 100
p = { 'alpha': 1, 'lb':[1.8,-0.5] }
a = 1
b = 10
# Inisialisasi - Generasi Pertama
cuckoos = gen_individu(n_individu,n_kota,a,b)
cuckoos_w_f = sort_individu(calculate_fitness(cuckoos,df))
cuckoos = cuckoos_w_f[;:-1]
new_cuckoos_w_f = np.copy(cuckoos_w_f)
new_cuckoos = np.copy(cuckoos)
# Main Program
while generasi<n_iter:
    #bangkitkan cuckoo secara acak dengan levy flight
    new_cuckoos = movement(new_cuckoos,p)
    #evaluasi fitness cuckoo
    new_cuckoos_w_f = sort_individu(calculate_fitness(new_cuckoos,df))
    # seleksi
    new_cuckoos = selec(new_cuckoos)

    #next generasi
    generasi = generasi+1

    # Print Best
    solusi(new_cuckoos_w_f)

```

```

n_iter = 100
generasi = 0
n_kota = len(df)
n_individu = 100
p = { 'alpha': 1, 'lb':[1.8,-0.5] }
a = 1
b = 10
# Inisialisasi - Generasi Pertama
cuckoos = gen_individu(n_individu,n_kota,a,b)
cuckoos_w_f = sort_individu(calculate_fitness(cuckoos,df))
cuckoos = cuckoos_w_f[;:-1]
new_cuckoos_w_f = np.copy(cuckoos_w_f)
new_cuckoos = np.copy(cuckoos)
# Main Program
while generasi<n_iter:
    #bangkitkan cuckoo secara acak dengan levy flight
    new_cuckoos = movement(new_cuckoos,p)
    #evaluasi fitness cuckoo
    new_cuckoos_w_f = sort_individu(calculate_fitness(new_cuckoos,df))
    # seleksi
    new_cuckoos = selec(new_cuckoos)

    #next generasi
    generasi = generasi+1

    # Print Best
    solusi(new_cuckoos_w_f)

```

Didalam inisialisasi parameter dimasukkan jumlah iterasi 100 dengan nilai generasi 0, n\_kota berjumlah kota yang ada dalam data "len(df)" dan n\_individu yang ingin digunakan sebanyak 100. Kemudian dibuat variabel p yang berisikan rumus bilangan parameter alpha dan lambda, dimana nilai lambda yang digunakan sekitar [1.8, -0.5] dengan variabel a dan b adalah nilai random, dimana plot berada antara a = 1 dan b = 10. Kemudian dilakukan inisialisasi dengan memasukkan generasi pertama, dengan fungsi gen\_individu yang berisikan rumus berupa fungsi dan parameter n\_individu, n\_kota, dimana a,dan b sebagai batas awal dan batas akhir nilai random. gen\_individu menggunakan fungsi lambda

dan akan digenerate oleh nilai random dari batas awal dan batas akhir. Kemudian akan di simpan menjadi variabel cuckoos

```
[ ] def diskritisasi(cuckoos):
    return np.argsort(cuckoos)

[ ] def calculate_fitness(cuckoos,df):
    d_cuckoos = diskritisasi(cuckoos)
    fitness = np.array( list(map(lambda x: calc_dist( x ,df.values), d_cuckoos )) )
    fitness = fitness.reshape( (-1,1) )
    return np.concatenate( ( cuckoos ,fitness ) ,axis=1)

[ ] def sort_individu(cuckoos_with_f):
    return cuckoos_with_f[cuckoos_with_f[:,1].argsort()]

[ ] cuckoos_w_f = sort_individu(calculate_fitness(cuckoos,df))

[ ] def solusi(cuckoos_w_f):

    df_kota = pd.DataFrame(diskritisasi(cuckoos_w_f[:, :-1]))
    cols = [ 'Urutan ' + str(i+1) for i in range( df_kota.shape[1] ) ]
    df_kota.columns = cols
    df_kota['Jarak'] = cuckoos_w_f[:, -1].reshape(-1,1)
    return df_kota

▶ solusi(cuckoos_w_f)
```

Kemudian dilakukan perhitungan nilai fitness dengan batas acak, lalu dilakukan sort data tersebut, sehingga data yang paling atas adalah data yang memiliki fitness terbaik "def calculate\_fitness(cuckoos, df)" kemudian di diskritisasi, variabel fitness akan menghitung distance, lalu akan di reshape dan akan di concatane atau di gabungkan antara array yang telah dibuat dan fitness yang telah dihitung di axis 1

```
▶ n_iter = 100
generasi = 0
n_kota = len(df)
n_individu = 100
p = { 'alpha': 1, 'lb':[1.8,-0.5] }
a = 1
b = 10
# Inisialisasi - Generasi Pertama
cuckoos = gen_individu(n_individu,n_kota,a,b)
cuckoos_w_f = sort_individu(calculate_fitness(cuckoos,df))
cuckoos = cuckoos_w_f[:, :-1]
new_cuckoos_w_f = np.copy(cuckoos_w_f)
new_cuckoos = np.copy(cuckoos)
# Main Program
while generasi<n_iter:
    #bangkitkan cuckoo secara acak dengan levy flight
    new_cuckoos = movement(new_cuckoos,p)
    #evaluasi fitness cuckoo
    new_cuckoos_w_f = sort_individu(calculate_fitness(new_cuckoos,df))
    # seleksi
    new_cuckoos = selec(new_cuckoos)

    #next generasi
    generasi = generasi+1

    # Print Best
    solusi(new_cuckoos_w_f)
```

Dilanjutkan ke Main Program, membangkitkan secara acak dengan levy flight. Kemudian fungsi akan melakukan evaluasi fitness cuckoo yang telah dibangkitkan, "new\_cuckoos\_w\_f" lalu dilakukan sort kembali. Dilakukanlah seleksi, dengan probabilitas apakah sarang nya ditemukan oleh pemilik sarang atau tidak, dengan menggunakan fungsi selec dengan bilangan random, lalu sesuai dengan array satu persatu diputuskan akan dibuang atau tidak, dan jika ada yg dibuang maka akan diisi dengan data random kembali. Lalu dikembalikan kembali data dari new\_cuckoos dengan data yang lebih baru lagi.

```
▶ n_kota = len(df)
n_individu = 15
a = 60
b = 400

gen_individu = lambda n_individu, n_kota,a,b: np.random.uniform(a,b,(n_individu,n_kota))
cuckoos = gen_individu(n_individu,n_kota,a,b)
cuckoos
```

```
[ ] def diskritisasi(cuckoos):
    return np.argsort(cuckoos)

[ ] def calculate_fitness(cuckoos,df):
    d_cuckoos = diskritisasi(cuckoos)
    fitness = np.array( [list(map(lambda x: calc_dist( x ,df.values), d_cuckoos ))] )
    fitness = fitness.reshape( (-1,1) )
    return np.concatenate( (cuckoos ,fitness) ,axis=1)

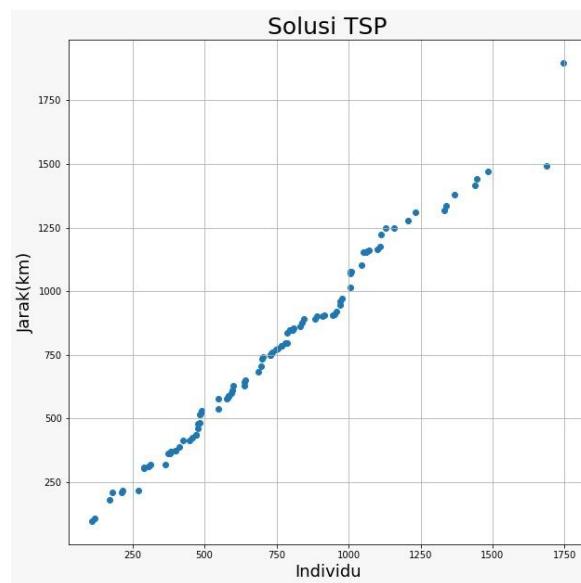
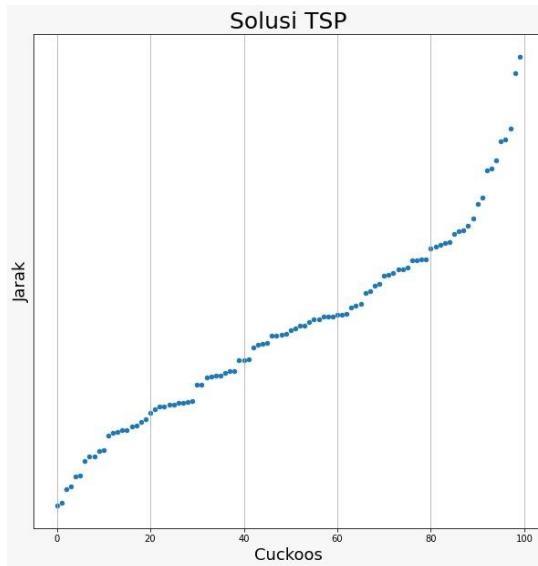
[ ] def sort_individu(cuckoos_with_f):
    return cuckoos_with_f[cuckoos_with_f[:,1].argsort()]

[ ] cuckoos_w_f = sort_individu(calculate_fitness(cuckoos,df))

[ ] def solusi(cuckoos_w_f):
    df_kota = pd.DataFrame(diskritisasi(cuckoos_w_f[:, :-1]))
    cols = [ 'Urutan ' + str(i+1) for i in range( df_kota.shape[1]-1 ) ]
    df_kota.columns = cols
    df_kota['Jarak'] = cuckoos_w_f[:, -1].reshape(-1,1)
    return df_kota

▶ solusi(cuckoos_w_f)
```

Visualisasi hubungan antara parameter dan solusi Terbaik. Parameter yang diobservasi : parameter metode , jumlah individu, maksimal iterasi



visualisasi solusi terhadap iterasi

	Urutan 1	Urutan 2	Urutan 3	Urutan 4	Urutan 5	Urutan 6	Urutan 7	Urutan 8	Jarak
0	1	3	2	5	0	4	8	7	109.0
1	0	4	1	2	5	6	3	7	120.0
2	4	0	6	2	7	1	5	3	169.0
3	5	3	6	0	4	7	2	1	180.0
4	6	7	4	1	2	5	0	3	214.0
...	...	...	...	...	...	...	...	...	...
95	6	4	0	2	5	7	3	1	1439.0
96	4	3	7	5	2	0	1	6	1445.0
97	3	4	2	0	7	5	6	1	1484.0
98	3	6	4	2	5	7	0	1	1687.0
99	2	3	1	7	5	4	6	0	1746.0

	Urutan 1	Urutan 2	Urutan 3	Urutan 4	Urutan 5	Urutan 6	Urutan 7	Urutan 8	Jarak
0	3	4	5	2	7	0	1	6	476.0
1	2	3	6	5	1	7	4	0	537.0
2	5	3	0	1	6	4	7	2	699.0
3	5	4	3	6	7	0	2	1	747.0
4	4	5	0	7	3	2	1	6	796.0
5	1	3	4	6	0	5	2	7	851.0
6	4	5	1	3	7	6	2	0	891.0
7	5	7	3	0	2	6	4	1	1066.0
8	1	0	5	3	7	6	4	2	1218.0
9	0	6	7	1	3	5	4	2	1256.0
10	2	1	3	6	7	5	0	4	1262.0
11	1	4	6	0	5	7	2	3	1322.0
12	1	3	5	7	6	2	4	0	1472.0
13	7	6	2	4	3	1	0	5	1521.0
14	3	2	0	1	7	5	6	4	1655.0

## CONCLUSION

Berdasarkan hasil dari percobaan algoritma cuckoo search dengan metode tsp didapatkan bahwa rute terpendek dengan jarak terbaik 109 KM adalah Bandung - Surabaya - Cirebon - Surakarta - Jakarta - Semarang - Yogyakarta - Malang

## REFERENCE

- Hardy, Wong, N. P., & Suwandy, D. (2013). *Penerapan Algoritma Cuuckoo Search Pada Travelling Salesman Problem*, 554-558.
- Ouaaraba, A., Ahiod, B., & Yang, X.-S. (2014). *Discrete Cuckoo Search Algorithm for the Traveling*, 7-8.

## APPENDIX

[https://colab.research.google.com/drive/1XR2PrdI3kWoM\\_IDKJ0kKav876ejoIeMx](https://colab.research.google.com/drive/1XR2PrdI3kWoM_IDKJ0kKav876ejoIeMx)