TUGAS ALGORITMA EVOLUSI

PENENTUAN JARAK TERPENDEK PADA JALUR DISTRIBUSI BARANG DI PULAU JAWA DENGAN MENGGUNAKAN ALGORITMA GENETIKA

Nama anggota kelompok :
Arya agung andika 165150207113010
Rama Humam Syarokha 165150207113001
Muhammad Asto Nugroho 165150207113012
Michael Eggi Bastian 165150218113006

Review paper

Judul jurnal : Penentuan Jarak terpendek pada jalur distribusi barang di pulau Jawa dengan menggunakan algoritma Genetika
Penulis : I Dewa Made Adi Baskara Joni & Vivine Nurcahyawati
Tahun : 2012
Reviewer : Arya Agung Andika, Rama Humam Syarokha, Muhammad Asto Nugrohom, Michael Eggi Bastian
Jurnal Nasional Pendidikan Teknik Informatika (JANAPATI)

Abstrak

Abstrak pada jurnal ini berisi tentang penjelasan permasalahan banyaknya kombinasi jalur distribusi barang melalui darat yang belum tentu memberikan solusi terbaik. Kemudian disebutkan juta solusi dari permasalahan ini yaitu menggunakan algoritma genetika. Menurut jurnal tersebut algoritma Genetika mampu memberikan hasil yang optimal dan diterapkan ke dalam suatu perangkat lunak.Penulis juga memberikan hasil pengujian dengan menggunakan kombinasi 5 kota sebagai tujuan distribusi. Hasil yang didapatkan dengan menggunakan PC & PM kurang dari 50 adalah 931 untuk tahap fitness inisialisasi dan 762 untuk fitness setelah dilakukan proses algoritma genetika

Abstraksi yang disajikan penulis hanya menggunakan bahasa Indonesia, menurut kami lebih baik apabila menggunakan 2 bahasa yaitu bahasa Indonesia dan bahasa Inggris. Secara keseluruhan isi dari abstraksi ini sudah lengkap yang mana mencakup permasalahan, solusi yang diharapkan , metode yang ingin digunakan dan hasil pengujian. Penjelasan yang diberikan juga langsung menuju ke topik bahasan, yang menurut kami sebagai pembaca menjadi mudah memahami jurnal ini.

Pendahuluan :

Pada paragraf pertama penulis menjelaskan tentang pentingnya distribusi di dalam bidang usaha. Penulis mengambil salah satu penelitian mengenai mencari jarak terpendek pada jalur distribusinya dalam hal transportasi laut. Sehingga mengacu pada jurnal tersebut penulis mencoba untuk

menerapkan algoritma genetika dalm jalur distribusi darat.

Pada paragraf selanjutnya dijelaskan secara singkat penegertian algoritma genetika dan manfaanya di dalam permasalahan ini.Disebutkan juga mengenai pembuatan perangkat lunak dengan masukan berupa kombinasi kota tujuan.

pada paragraf tiga menjelaskan apa tujuan dari penelitian ini yaitu untuk menantukan jalur distribusi barang yang terbaik dengan menggunakan algoritma genetika. Dalam penentuan jalur tersebut difokuskan pada pendistribusian barang ke agen - agen yang melalui jalur darat. Jalur pada penelitian ini yang digunakan hanya berada di pulau Jawa dan kota awal merupakan kota tujuan akhir dari jalur distribusi.

secara keseluruhan pendahuluan yang disajikan sudah cukup baik namun pada latar belakang permasalahan tidak dijelaskan secara lebih mendalam. Penulis juga tidak mengungkapkan mengenai penelitian - penelitian lainnya yang berkaitan dengan permasalahan ini.Hal ini penting untuk diungkapkan sebagai landasan bahwa penelitian ini layak untuk dilakukan.

Dasar Teori :

Pada bagian dasar teori Penulis membagi ke dalam 2 sub bagian yaitu penjelasan mengenai teori algoritma Genetika dan penjelasan mengenai teori Algortima Genetika *Canonical*. Pada pengertian algoritma genetika dijelaskan bahwa algortima genetika adalah algoritma pencarian heuristik yang didasarkan atas mekanisme evolusi biologis.Algoritma genetika menawarkan suatu solusi pemecahan masalah yang terbaik, dengan memanfaatkan metode seleksi, *crossover*, dan mutasi. Pengertian algoritma genetika *Canonical* adalah salah satu model GA dengan ciri representasi biner dari solusi individu, masalah sederhana *crossover* bebas, operator mutasi dan aturan pemilihan proposional.

penjelasan teori yang diberikan tidak sepenuhnya lengkap, peneliti tidak menjelaskan dasar teori mengenai metode seleksi,crossover, dan mutasi di bab ini. Namun sebaliknya penulis menjelaskan metode seleksi,crossover dan mutasi di bab 3 yaitu Implementasi perangkat lunak.Menurut kami sebaiknya penjelasan teori - teori ini dijelaskan pada bagian bab dasar teori.

Implementasi perangkat lunak

Pada penjelasan bab ini penulis membagi ke dalam beberapa sub bab antara lain: tahap encoding, proses seleksi, proses rekombinasi/crossover dan proses mutasi. tahapan - tahapan algoritma genetika yang digunakan dalam jurnal ini dapat dilihat pada gambar dibawah :

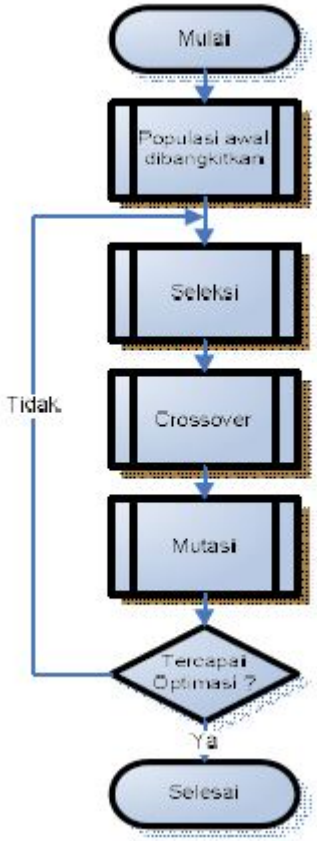
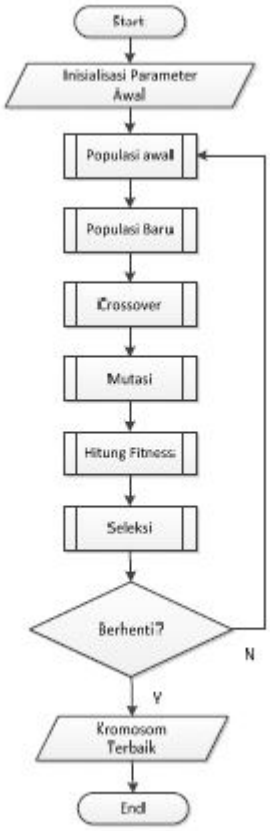


Diagram alir yang digambarkan diatas diketahui bahwa proses seleksi dilakukan setelah inisialisasi populasi kemudian baru melakukan reproduksi (crossover dan mutasi). Dari beberapa jurnal yang kami baca memang banyak yang menerapkan proses seleksi terlebih dahulu dibandingkan reproduksi seperti pada jurnal penerapan algoritma genetika pada permasalahan matematika oleh Awang Andhyka(2018) dan jurnal optimalisasi penjadwalan produksi dengan metode algoritma genetika di PT.Progress Diecast oleh Lily Amelia & Aprianto (2011). Namun dari referensi yang kami temukan seperti pada modul pembelajaran yang dibuat oleh wayan firdaus Mahmudy(2015) dan beberapa jurnal seperti penjadwalan kapal penyeberangan oleh Ria Febriyana, Wayan Firdaus Mahmudy (2016) jurnal implementasi algoritma genetika untuk penjadwalan *Customer service* oleh Damayanti, Madi Putri, M.Ali Fauzi(2017) menyatakan bahwa tahapan algoritma dimulai dari inisialisasi populasi, crossover, mutasi, dan seleksi. Hal ini bertujuan untuk mendapatkan individu terbaik dari hasil reproduksi yang siap digunakan untuk generasi selanjutnya. Maka dari itu kelompok kami nantinya akan menggunakan alur tahap ini untuk diterapkan di dalam implementasi jurnal penentuan jarak terpendek pada jalur distribusi di pulau Jawa. Berikut adalah alur yang akan kami gunakan dalam implementasi koding



Teknik Encoding

encoding merupakan fungsi pengkodean yang digunakan untuk merepresentasikan pemetaan variabel objek ke dalam bentuk string. Menurut jurnal *Encoding Schemes in Genetic Algorithm* oleh Anit Kumar(2013) teknik encoding sendiri ada beberapa jenis yaitu *binary encoding, octal encoding, hexadecimal encoding,permutation encoding, value encoding dan tree encoding.* Pada penelitian ini teknik encoding yang digunakan adalah permutation encoding. Permutation encoding adalah kumpulan - kumpulan angka yang mewakili posisi dalam sebuah rangkaian. Contoh representasi nya adalah seperti 1,2,5,4,3. Angka 1,2,5,4,3 adalah angka yang mewakili nama kota pada jalur distribusi.

Menurut kami penggunaan teknik *permutation encoding* pada jurnal ini sudah tepat karena disebutkan pada jurnal *Encoding Schemes in Genetic Algorithm* bahwa cocok digunakan pada permasalahan *travel salesman problem* dimana angka - angka akan direpresentasikan sebagai kota. Data kota - kota di pulau jawa beserta jaraknya dapat dilihat pada gambar dibawah.

DAFTAR JARAK ANTAR KOTA DI PULAU JAWA

	MADIUN	MAGELANG	MALANG	PEKALONGAN	PURWOREJO	PROBOLINGGO	REMBANG	SEMARANG	SERANG	SUKABUMI	SURABAYA	SOLO	TASIKMALAYA	TEGAL	CILACAP	WONOSOBO
Bandung	581	403	764	266	362	774	476	367	258	96	675	467	106	202	259	339
Banyuwangi	412	633	281	696	637	190	490	597	1171	1060	289	506	893	762	768	678
Jakarta	699	522	882	384	504	892	594	485	89	115	793	585	275	320	428	481
Bondowoso	315	517	184	601	540	93	393	500	1074	965	192	409	796	665	671	560
Bogor	707	530	890	392	488	900	602	493	143	61	801	593	232	328	385	465
Cirebon	451	274	634	136	256	644	346	237	337	226	545	337	121	72	203	233
Jember	320	522	185	607	545	99	399	506	1090	968	198	414	801	671	695	565
Jogjakarta	179	43	363	181	66	381	212	118	654	524	327	65	322	245	197	107
Kediri	61	283	109	376	306	177	206	295	868	758	124	175	562	460	437	326
Madiun		221	184	315	245	222	145	214	788	677	169	114	501	379	375	265
Magelang	221		406	136	44	443	184	75	611	500	370	108	297	202	174	64
Malang	184	406		499	429	91	275	397	671	860	89	298	682	562	560	449
Pekalongan	315	138	499		170	508	210	101	473	362	409	201	257	64	195	134
Purworejo	245	44	429	170		446	228	119	593	458	393	131	256	223	131	54
Probolinggo	222	443	91	508	445		300	407	981	870	99	316	702	572	598	487
Rembang	145	184	275	210	228	300		109	683	572	201	147	467	274	355	228
Semarang	214	75	397	101	119	407	109		574	463	308	100	358	165	246	119
Serang	780	611	971	473	593	991	693	574		203	882	674	364	409	517	570
Sukabumi	677	500	860	362	458	870	572	463	203		771	563	202	298	355	435
Surabaya	169	370	89	409	393	99	201	308	882	771		262	649	473	523	434
Solo	114	108	298	201	131	316	147	100	674	563	282		387	265	262	151
Tasikmalaya	501	297	682	257	256	702	467	358	364	202	649	387		193	153	233
Tegal	379	202	562	64	223	572	274	165	409	298	473	265	193		131	197
Cilacap	375	174	560	195	131	598	355	246	517	355	523	262	153	131		127
Wonosobo	265	64	449	134	54	487	228	119	570	435	434	151	233	197	127	

Gambar 3. Jarak antar Kota di Jawa

Proses Seleksi

Proses seleksi bertanggung jawab untuk melakukan pemilihan terhadap individu yang hendak diikutkan dalam proses reproduksi (wayan firdaus mahmudy & candra dewi.2014).Pada bagian ini penulis menjelaskan teknik seleksi yang digunakan beserta pengertiannya. Teknik seleksi yang digunakan adalah *Roulette wheel.*Metode Roulette Wheel Selection merupakan sebuah pendekatan yang dilakukan dengan menghitung nilai probabilitas seleksi (prob) tiap individu berdasarkan nilai fitnessnya, nilai prob berfungsi sebagai roullete wheelnya. Dari nilai probabilitas seleksi akan dihitung nilai probabilitas kumulatif (probCum) yang nantinya akan digunakan untuk melakukan seleksi pada tiap individu, nilai probCum berfungsi sebagai pemutar roullete wheelnya. Metode Roulette Wheel Selection bertujuan untuk memberikan peluang yang lebih besar terhadap individu yang memiliki nilai fitness yang lebih besar untuk terpilih dan dipertahankan pada generasi selanjutnya. Akan tetapi karena berupa peluang maka tidak menjamin bahwa individu terbaik akan selalu terpilih untuk masuk pada generasi selanjutnya.

Berikut adalah contoh proses seleksi roulette wheel, Namun contoh proses seleksi ini terdapat perbedaan dibandingkan dengan paper ini, pada bagian fitness dimana jika di paper fungsi cost disebut dengan fitness, maka fitness pada paper akan diganti menjadi fungsi cost dan inverse pada paper digunakan sebagai fungsi fitness.

1.)Menentukan Fungsi cost

fungsi cost yang digunakan adalah jarak antar kota

Kromosom[1] : [Malang – Rembang – Solo – Semarang] atau [B D E C]
Kromosom[2] : [Rembang – Malang – Solo - Semarang] atau [D B E C]
Kromosom[3] : [Semarang – Malang – Rembang - Solo] atau [C B D E]
Kromosom[4] : [Solo – Malang – Semarang - Rembang] atau [E B C D]
Kromosom[5] : [Solo – Semarang – Malang - Rembang] atau [E C B D]
Kromosom[6] : [Semarang – Rembang – Solo - Malang] atau [C D E B]

cost[1]	AB + BD + DE + EC + CA	920
cost[2]	AD + DB + BE + EC + CA	1174
cost[3]	AC + CB + BD + DE + EA	1289
cost[4]	AE + EB + BC + CD + DA	1185
cost[5]	AE + EC + CB + BD + DA	1153
cost[6]	AC + CD + DE + EB + BA	952

2.) Menentukan fungsi fitness dan menghitung fitness

$$fitness[i] = \frac{1}{cost[i]}$$

Fitness[1] = 1/920 = 0.001087
Fitness[2] = 1/1174 = 0.000852
Fitness[3] = 1/1289 = 0.000776
Fitness[4] = 1/1185 = 0.00084
Fitness[5] = 1/1153 = 0.000867
Fitness[6] = 1/952 = 0.00105

3.) Menghitung probabilitas kemungkinan terpilihnya individu

$$P[i] = \frac{fitness[i]}{totalFitness}$$

P[1] = 0,001087 / 0,005476 = 0,199
P[2] = 0,000852 / 0,005476 = 0,156
P[3] = 0,000776 / 0,005476 = 0,142
P[4] = 0,000844 / 0,005476 = 0,154
P[5] = 0,000867 / 0,005476 = 0,158
P[6] = 0,00105 / 0,005476 = 0,192

4.) Menentukan probabilitas kumulatif

C[1] = 0,199
C[2] = 0,199 + 0,156 = 0,355
C[3] = 0,355 + 0,142= 0,497
C[4] = 0,497 + 0,154 = 0,651
C[5] = 0,651 + 0,158 = 0,809
C[6] = 0,809 + 0,192 = 1,001

5.) Membangkitkan bilangan random dengan jarak 0 sampai 1 sebanyak jumlah individu dalam populasi

R[1] = 0,314
R[2] = 0,111
R[3] = 0,342
R[4] = 0,743
R[5] = 0,521
R[6] = 0,411

6.) Membandingkan probCum individu yang terpilih dengan bilangan r yang telah dibangkitkan. Jika r kurang dari atau sama dengan probCum maka individu tersebut akan terpilih dan dipertahankan untuk generasi selanjutnya.Maka setelah perbandingan, hasil individu yang terpilih adalah:

Kromosom[1] : [2] :[Rembang – Malang – Solo - Semarang] atau [D B E C]
Kromosom[2] :[1] :[Malang – Rembang – Solo – Semarang] atau [B D E C]
Kromosom[3] : [3] :[Semarang – Malang – Rembang - Solo] atau [C B D E]
Kromosom[4] : [5] : [Solo – Semarang – Malang - Rembang] atau [E C B D]
Kromosom[5] : [4] :[Solo – Malang – Semarang - Rembang] atau [E B C D]
Kromosom[6] : [6] : [Semarang – Rembang – Solo - Malang] atau [C D E B]

Proses rekombinasi/crossover

Crossover merupakan proses perkawilan silang untuk menghasilkan keturunan dari dua buah kromosom induk yang terpilih(wayan firdaus mahmudy & candra dewi.2014).Pada bagian ini penulis menjelaskan mengenai teknik *crossover* dan cara menerapkannya pada permasalahan ini. Teknik *crossover* yang digunakan dalam penelitian ini adalah teknik *Order crossover (OX)* diperkenalkan oleh Davis Teknik ini diawali dengan membangkitkan dua bilangan acak. Kemudian gen yang berada diantara kedua bilangan acak akan disalin ke offspring dengan posisi yang sama. Langkah berikutnya untuk mendapatkan offspring pertama adalah mengurutkan gen yang berada pada parent kedua dengan urutan gen yang berada pada posisi setelah bilangan acak kedua diikuti dengan gen yang berada pada posisi sebelum bilangan acak pertama dan diakhiri dengan gen yang berada pada posisi diantara kedua bilangan acak. Kemudian gen yang telah diurutkan tersebut dibandingkan dengan offspring pertama. Apabila gen tersebut ada pada offspring kedua maka abaikan gen tersebut dari urutan itu. Kemudian masukkan urutan yang baru saja didapat pada offspring dengan cara memasukkan urutan gen pada posisi setelah bilangan acak kedua terlebih dahulu dan sisanya dimasukkan pada posisi sebelum bilangan acak pertama. Begitu juga untuk menghasikan offspring kedua. Berikut adalah contoh cara menghitung *crossover* untuk kasus ini.

1.) tentukan cr(*crossover rate*) dan bilangan random sebanyak n individu dalam populasi

R[1]	0.541
R[2]	0.211
R[3]	0.302
R[4]	0.877
R[5]	0.771
R[6]	0.131

2.) Diasumsikan pc = 50% atau 0.5. kemudian dicari dengan kategori kromosom ke-K yang dipilih sebagai induk jika R[K] kurang dari pc. maka dari gambar diatas yang terpilih adalah kromosom[2],kromosom[3],kromosom[6]. Proses selanjutnya menentukan posisi crossover dengan membangkitkan bilangan acak antara 1 - (panjang kromosom -1)

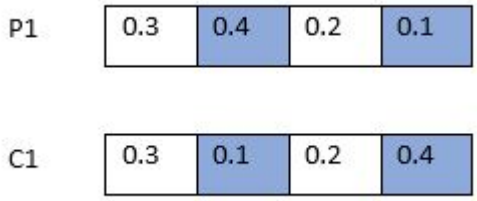
C[2] = 2
C[3] = 1
C[6] = 2

3.)kemudian lakukan proses *crossover* seperti pada gambar dibawah

Proses *crossover* :
Kromosom[2] = Kromosom[2] >< Kromosom[3] = [B **D** E C] >< [C B D E] = [B **D** C E]
Kromosom[3] = Kromosom[3] >< Kromosom[6] = [**C** B D E] >< [C D E B] = [**C** D E B]
Kromosom[6] = Kromosom[6] >< Kromosom[2] = [C **D** E B] >< [B D E C] = [C **D** B E]

Proses Mutasi

Proses mutasi berguna untuk menciptakan individu baru dengan melakukan modifikasi terhadap satu atau lebih gen dalam individu yang sama (wayan firdaus mahmudy & candra dewi.2014). teknik mutasi yang digunakan dalam jurnal ini adalah *swapping mutation* atau *Reciprocal Random Mutation* dimana cara kerja metode ini yaitu dengan memilih dua posisi (exchange point / XP) secara random di kromosom induk kemudian menukarkan nilai pada posisi tersebut (Mahmudy , 2015). Berikut adalah contoh *Reciprocal Exchange Mutation*



Kodingan dibawah ini akan mengimplementasikan percobaan yang dilakukan di dalam jurnal PENENTUAN JARAK TERPENDEK PADA JALUR DISTRIBUSI BARANG DI PULAU JAWA DENGAN MENGGUNAKAN ALGORITMA GENETIKA. Terdapat beberapa perbedaan dalam proses yang akan diimplementasikan, dimana perbedaan tersebut meliputi susunan langkah langkah algoritma genetika, fungsi fitness, dan jumlah kota yang akan digunakan. Perbedaan - perbedaan ini telah dijelaskan pada bagian sebelumnya. Percobaan ini akan menggunakan teknik order crossover, reciprocal mutation/swap mutation dan teknik seleksi roulette wheel.

In [2]:
```
#import library yang dibutuhkan
import numpy as np
import pandas as pd
import random as rd
```

In [6]:
```
#ambi dataset yang telah dibuat sebelumnya
dataset = pd.read_excel('datasetAlev.xlsx')
#jarak kota kediri ke kota lain dipisah karena kota ini tidak dirandom dan agar m
jarak_kota = dataset.iloc[1:, :].values

#jarak antar kota satu ke kota lainnya
jarak_kediri = dataset.iloc[0:1,:].values

print(dataset)
print("\n Jarak antar kota")
print('\n',jarak_kota)
print("\n Jarak antar kota ke kota kediri")
print('\n',jarak_kediri)
```

```
             Madiun  Magelang  malang  pekalongan  purworejo  probolinggo  \
kediri           81       233     103         376        306          177
madiun            0       221     184         315        245          222
magelang        221         0     406         138         44          443
malang          184       406       0         499        429           91
pekalongan      315       138     499           0        170          508
purworejo       245        44     429         170          0          446
probolinggo     222       443      91         508        446            0
rembang         145       184     275         210        228          300
semarang        214        75     397         101        119          407
serang          788       611     671         473        593          981
sukabumi        677       500     860         362        458          870
surakarta       169       370      89         409        393           99
solo            114       108     298         201        131          316
tasikmalaya     501       297     682         257        256          702
tegal           379       202     562          64        223          572
cilacap         375       174     560         195        131          598
wonosobo        265        64     449          34         54          487

             rembang  semarang  serang   sukabumi  surakarta   solo  \
kediri           206       295      869       758         124   175
madiun           145       214      788       677         169   114
magelang         184        75      611       500         370   108
malang           275       397      671       860          89   298
pekalongan       210       101      473       362         409   201
purworejo        228       119      593       458         393   131
probolinggo      300       407      981       870          99   316
rembang            0       109      683       572         201   147
semarang         109         0      574       463         308   100
serang           683       574        0       203         882   674
sukabumi         572       463      203         0         771   563
surakarta        201       308      882       771           0   262
solo             147       100      674       563         262     0
tasikmalaya      467       358      364       202         649   387
tegal            274       165      409       298         473   265
cilacap          355       246      517       355         523   262
wonosobo         228       119      570       435         434   151

             tasikmalaya  tegal   cilacap  wonosobo
kediri               562     460      437       326
madiun               501     379      375       265
magelang             297     202      174        64
malang               682     562      560       449
```

```
pekalongan          257      64      195        34
purworejo           256     223      131        54
probolinggo         702     572      598       487
rembang             467     274      355       228
semarang            358     165      246       119
serang              364     409      517       570
sukabumi            202     298      355       435
surakarta           649     473      523       434
solo                387     265      262       151
tasikmalaya           0     193      153       233
tegal               193       0      131       197
cilacap             153     131        0       127
wonosobo            233     197      127         0
```

```
 Jarak antar kota

 [[  0 221 184 315 245 222 145 214 788 677 169 114 501 379 375 265]
 [221   0 406 138  44 443 184  75 611 500 370 108 297 202 174  64]
 [184 406   0 499 429  91 275 397 671 860  89 298 682 562 560 449]
 [315 138 499   0 170 508 210 101 473 362 409 201 257  64 195  34]
 [245  44 429 170   0 446 228 119 593 458 393 131 256 223 131  54]
 [222 443  91 508 446   0 300 407 981 870  99 316 702 572 598 487]
 [145 184 275 210 228 300   0 109 683 572 201 147 467 274 355 228]
 [214  75 397 101 119 407 109   0 574 463 308 100 358 165 246 119]
 [788 611 671 473 593 981 683 574   0 203 882 674 364 409 517 570]
 [677 500 860 362 458 870 572 463 203   0 771 563 202 298 355 435]
 [169 370  89 409 393  99 201 308 882 771   0 262 649 473 523 434]
 [114 108 298 201 131 316 147 100 674 563 262   0 387 265 262 151]
 [501 297 682 257 256 702 467 358 364 202 649 387   0 193 153 233]
 [379 202 562  64 223 572 274 165 409 298 473 265 193   0 131 197]
 [375 174 560 195 131 598 355 246 517 355 523 262 153 131   0 127]
 [265  64 449  34  54 487 228 119 570 435 434 151 233 197 127   0]]

 Jarak antar kota ke kota kediri

 [[ 81 233 103 376 306 177 206 295 869 758 124 175 562 460 437 326]]
```

In [8]:
```
#inisialisasi inidividu dengan rentang 0 - 15 sesua dengan jumlah kota yang ada s
def individu(panjang_gen):
    #buat sebuat dengan rentang 0 sampai panjang_gen
    arr = list(range(0, panjang_gen))
    #lakukan permutasi pada array arr
    x = np.random.permutation(range(0, panjang_gen))
    #masukkan hasil permutasi ke dalam list
    x.tolist()
    return x
```

In [9]:
```
individu(16)
```

Out[9]:
```
array([ 6,  1, 14,  3,  0, 15, 10,  8,  5,  7, 12, 13,  2,  9,  4, 11])
```

In [10]:
```
#inisialisasi populasi awal dengan parameter panjang gen dan ukuran populasi
def populasi(panjang_gen,size):
    #inisialisasi array baru untuk menyimpan populasi
    pops1 = []
    #looping sesuai dengan ukuran populasi
    for i in range(size):
        #masukkan individu ke dalam array populasi
        pops1.append(individu(panjang_gen))
    return pops1
```

In [19]:
```
pop_awal = populasi(16,15)
for i in range(15):
    print(pop_awal[i])
```

```
[11 12  8  3  4 14  7 15  0  2  1  9 10  5 13  6]
[ 3  4 14 13 11 15  2 12  0  7  5 10  6  8  9  1]
[10 12  3  8  4 14  6  1  5  0  7 13 11 15  2  9]
[ 5  7 12  3 13  4 11  2 14  0  1 10 15  8  6  9]
[14 13  1 10  5  6 12  2  4  0 11  9  3  8 15  7]
[11  3 10  5  4 12 15  0  8  9  2  1  7 13  6 14]
[14  8  0  2  5 11 12  7  1  9  4 15  6 10  3 13]
[ 1 10  0  5  8 14 11  7 13  2 12 15  6  3  9  4]
[10  7 12 14  6  5  8 13 11  9  0  3  4 15  2  1]
[11  1 10 12  3  5  0 14 15  6  4 13  7  8  2  9]
[ 0  1  4 13  2 14  5 11 12  6 15 10  9  3  7  8]
[ 6  9 13 10 11 12  3 14  7  5  4  1 15  2  8  0]
[14  4  8  6 10 13  3 12  5  0  9  2 11  1  7 15]
[ 8  3  6 14  7 15  2 12  4 11  1  9  0 10 13  5]
[ 9  6 14  8  7  2 15 13 12 10  1  4  3  0 11  5]
```

In [17]:
```
#crossover dengan menggunakan teknik order crossiver
#parameter yang diperlukan adalah crossover rate (cr) dan populasi
def crossover(cr,populasi):
    #buat sebuah array kosong
    hasilCros = []
    #variabel pops digunakan untuk menyimpan array hasil tahap populasi
    pops = populasi
    #menentukan jumlah offspring sesuai dengan cr yang dimasukkan dikali dengan p
    offspring = int(cr * len(pops))
    #melakukan permutasi secara random dengan rentang 0 - panjang populasi
    permut1 = np.random.permutation(range(0, len(pops)))
    #inisialisasi bilangan acak permutasi sebanyak nilai offspring
    #ini dilakukan untuk mendapatkan index pada populasi yang akan dilakukan cros
    #cross1 akan berperan sebagai parent1
    cross1 = permut1[0:offspring]
    #cross2 merupakan hasil acak dari cross1
    #cross2 akan digunakan sebagai parent ke 2 untuk dipasangkan dengan parent1/c
    cross2 = sorted(cross1,key=lambda k:np.random.random())
    #variabel random digunakan untuk menentukan titik crossover di dalam suatu in
    random = np.random.randint(0,len(pops),len(cross2))

    #buat variabel yang isinya sama dengan pops
    uji = pops.copy()
    #lakukan looping sepanjang cross1
    for i in range(len(cross1)):
        #masukkan nilai individu yang sudah dipotong pada titik tertentu
        hasilCros.append(pops[cross1[i]][:random[i]+1])
    #masukkan nilai hasilCros kedalam variabel baru
    hasilCrossBaru = hasilCros.copy()

    #perulangan ini dilakukan untuk memasukkan nilai dari parent2 yang belum ada
    for j in range(len(cross1)):
        for k in range(len(uji[0])):
            if(uji[cross2[j]][k] not in hasilCrossBaru[j]):
                hasilCrossBaru[j] = np.append(hasilCrossBaru[j], uji[cross2[j]][k

    #setelah hasil offspring didapatkan maka offspring tersebut dimasukkan ke dal
    for i in range(len(cross1)):
        uji.append(hasilCrossBaru[i])
    #print(random)
    #print('\n cross1 \n', cross1 , '\n')
    #print('\n cross2 \n',cross2 , '\n')
    #print('sebelum crossover')
    #for i in range(len(hasilCrossBaru)):
    #    print(hasilCrossBaru[i])
    #print('\n hasil sebelum crossover')
    return(uji)
```

In [20]:
```
#hasil populasi setelah di crossover
crossing = crossover(0.4,pop_awal)
for i in range(len(crossing)):
    print(crossing[i])
```

```
[11 12  8  3  4 14  7 15  0  2  1  9 10  5 13  6]
[ 3  4 14 13 11 15  2 12  0  7  5 10  6  8  9  1]
[10 12  3  8  4 14  6  1  5  0  7 13 11 15  2  9]
[ 5  7 12  3 13  4 11  2 14  0  1 10 15  8  6  9]
[14 13  1 10  5  6 12  2  4  0 11  9  3  8 15  7]
[11  3 10  5  4 12 15  0  8  9  2  1  7 13  6 14]
[14  8  0  2  5 11 12  7  1  9  4 15  6 10  3 13]
[ 1 10  0  5  8 14 11  7 13  2 12 15  6  3  9  4]
[10  7 12 14  6  5  8 13 11  9  0  3  4 15  2  1]
[11  1 10 12  3  5  0 14 15  6  4 13  7  8  2  9]
[ 0  1  4 13  2 14  5 11 12  6 15 10  9  3  7  8]
[ 6  9 13 10 11 12  3 14  7  5  4  1 15  2  8  0]
[14  4  8  6 10 13  3 12  5  0  9  2 11  1  7 15]
[ 8  3  6 14  7 15  2 12  4 11  1  9  0 10 13  5]
[ 9  6 14  8  7  2 15 13 12 10  1  4  3  0 11  5]
[ 8  3  6 14  7 15  2 12  4 11  1  9  0 10 13  5]
[ 6  9 13 10 11 12  3 14  7  5  4  1 15  2  8  0]
[ 9  6 14  8  7  2 15 13 12 10  1  4  3  0  5 11]
[10  7 12 14  6  5  9  8  2 15 13  1  4  3  0 11]
[ 3  4 14 13 11 15  2 12  0  7  5 10  6  8  9  1]
[ 1 10  0  5  8 14 11  7 13  2 12 15  6  3  9  4]
```

In [21]:
```
#mutasi dengan menggunakan teknik reciprocal mutation
#parameter yang diperlukan adalah mutation rate(mr), populasi hasil crossover, da
def mutasi(mr,hasilCros,popSize):
    #copy populasi hasil crossover ke dalam variabel
    test = hasilCros.copy()
    #tentukan jumlah individu yang ingin dimutasi
    jumlah_gen_mut = int(mr * popSize)
    #variabel randi digunakan untuk mentukan nilai permutasi random dari 0 sampai
    #randi akan dipakai untuk menentukan individu yang akan dimutasi
    randis = np.random.permutation(range(0, len(test)))
    #variabel random akan membuat sebuah array sebanyak jumlah individu yang ingi
    random = randis[0:jumlah_gen_mut]


    forMutasi = test.copy()
    for i in range(len(random)):
        permut1 = np.random.permutation(range(0, len(test[0])))
        randi =permut1[0:2]
        #membuat variabel temp sebagai array tampungan sementara
        temp = []
        #tambahkan gen berdasarkan random dan randi
        temp.append(test[random[i]][randi[0]])
        temp.append(test[random[i]][randi[1]])
        #individu pada array formutasi dengan indeks random ke-i akan dihapus ter
        forMutasi[random[i]] = np.delete(forMutasi[random[i]],randi[0],None)
        forMutasi[random[i]] = np.insert(forMutasi[random[i]],randi[0],temp[1],No
        forMutasi[random[i]] = np.delete(forMutasi[random[i]],randi[1],None)
        forMutasi[random[i]] = np.insert(forMutasi[random[i]],randi[1],temp[0],No
        #print(randi[0])
        #print(randi[1])
    #print(randi)
    #untuk setiap offspring hasil mutasi ditambahkan ke dalam populasi hasil cros
    for i in range (len(random)):
        test.append( forMutasi[random[i]])

    return test
```

In [23]:
```
mut = mutasi(0.2,crossing,15)
for i in range(len(mut)):
    print(mut[i])
```

```
[ 5  7 12  3 13  4 11  2 14  0  1 10 15  8  6  9]
[14 13  1 10  5  6 12  2  4  0 11  9  3  8 15  7]
[11  3 10  5  4 12 15  0  8  9  2  1  7 13  6 14]
[14  8  0  2  5 11 12  7  1  9  4 15  6 10  3 13]
[ 1 10  0  5  8 14 11  7 13  2 12 15  6  3  9  4]
[10  7 12 14  6  5  8 13 11  9  0  3  4 15  2  1]
[11  1 10 12  3  5  0 14 15  6  4 13  7  8  2  9]
[ 0  1  4 13  2 14  5 11 12  6 15 10  9  3  7  8]
[ 6  9 13 10 11 12  3 14  7  5  4  1 15  2  8  0]
[14  4  8  6 10 13  3 12  5  0  9  2 11  1  7 15]
[ 8  3  6 14  7 15  2 12  4 11  1  9  0 10 13  5]
[ 9  6 14  8  7  2 15 13 12 10  1  4  3  0 11  5]
[ 8  3  6 14  7 15  2 12  4 11  1  9  0 10 13  5]
[ 6  9 13 10 11 12  3 14  7  5  4  1 15  2  8  0]
[ 9  6 14  8  7  2 15 13 12 10  1  4  3  0  5 11]
[10  7 12 14  6  5  9  8  2 15 13  1  4  3  0 11]
[ 3  4 14 13 11 15  2 12  0  7  5 10  6  8  9  1]
[ 1 10  0  5  8 14 11  7 13  2 12 15  6  3  9  4]
[ 8  3  6 11  7 15  2 12  4 14  1  9  0 10 13  5]
[14 13  1 10  5  6 12  2  4  0 11  9  8  3 15  7]
```

In [24]:
```
#menghitung fitness setiap individu dalam populasi hasil mutasi
def fitness(populasi):
    #variabel pops menampung isi array dari populasi yang direturn dari tahap muta
    pops = populasi
    #buat sebuah array dengan nilai nol sepanjang pops
    cost = np.zeros((len(pops),1))
    fitness = np.zeros((len(pops),1))
    #perulangan ini digunakan untuk mendapatkan nilai jarak antar kota lalu kemud
    for i in range(len(pops)):
        temp = 0
        for j in range (len(pops[i])):
            if(j != len(pops[i])-1):
                #menghitung jarak antar kota setiap individu
                temp = temp + jarak_kota[pops[i][j]][pops[i][j+1]]
                cost[i] = temp
        #menambahkan jarak kota dari kediri ke kota lain dan kota lain ke kota ked
        cost[i] = cost[i] + jarak_kediri[0][pops[i][0]] + jarak_kediri[0][pops[i]
        #menghitung fitness
        #rumus fitness untuk kasus ini adalah 1/cost
        fitness[i] = 1 / (cost[i])
    return fitness
```

In [25]:
```
#contoh hasil fitness
new_fitness = fitness(mut)
for i in range(len(new_fitness)):
    print(new_fitness[i])
```

```
[ 0.0001872]
[ 0.00018532]
[ 0.00015891]
[ 0.00015485]
[ 0.00017071]
[ 0.00017709]
[ 0.00018093]
[ 0.00016502]
[ 0.0001634]
[ 0.00015389]
[ 0.0001471]
[ 0.00017106]
[ 0.00016062]
[ 0.00015466]
[ 0.00016215]
[ 0.00015466]
[ 0.00017106]
[ 0.00015941]
[ 0.00019968]
[ 0.00018532]
[ 0.00016502]
[ 0.00016186]
[ 0.00019369]
[ 0.00018255]
```

In [26]:
```
#pada tahap seleksi seleksi yang digunakan adalah metode roulette wheel
#parameter yagn dibutuhkan adalah jumlah_kota, fitness hasil mutasi, populasi has
def seleksi_roulette(high,fitness,hasil_mutasi,size):
    #buat sebuah array kosong
    newpops2 = []
    #tamuung nilai fitness dan hasil_mutasi ke dalam variabel tertentu
    inverse = fitness
    pops = hasil_mutasi
    #hitung nilai total_q dengan menjumlahkan seluruh nilai fitness
    total_q = sum(inverse)
    #buat sebuah array dengan nilai 0 sebanyak panjang inverse
    prob  = np.zeros((len(inverse),1))
    kumulatif = np.zeros((len(inverse),1))
    #menghitung nilai kumulatif dengan menambahkan nilai probablitas saat ini deng
    for i in range (len(inverse)):
        prob[i] = inverse[i] / total_q
        if( i == 0 ):
            kumulatif[i] = prob[i]
        else:
            kumulatif[i] = kumulatif[i - 1] + prob[i]
    #akan membangkitkan nilai random sebanyak ukuran populasi
    random = np.random.rand(size,1)
    newpops = []
    #perulangan ini digunakan untuk membandingkan kumulatif ke-p dengan nilai ran
    #nilai kumulatif tersebut maka populasi ke-p akan dimasukkan ke dalam array p
    for i in range(size):
        p = 0
        for j in range(len(inverse)):
            if(random[i] < kumulatif[p] and p != size):
                newpops.append(p)
                newpops2.append(pops[p])
                break
            else:
                p = p + 1


    #print('ini inverse \n',inverse,'\n')
    #print('ini total_q',total_q)
    #print('prob \n',prob,'\n')
    #print('kumulatif \n',kumulatif,'\n')
    #print('random \n',random,'\n')
    #print(newpops,'\n')
    return(newpops2)
```

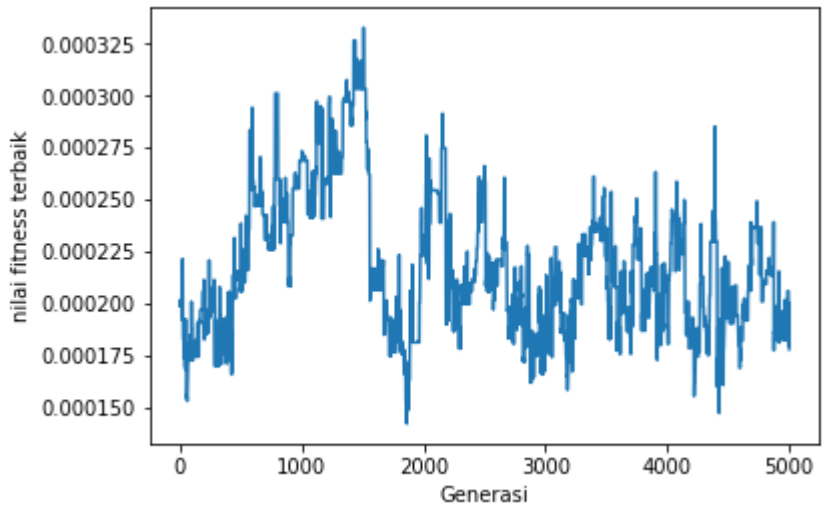
In [27]:
```
#berikut adalah contoh populasi hasil seleksi
seleksi_roulette(16,new_fitness,mut,10)
```

Out[27]:
```
[array([ 3,  4, 14, 13, 11, 15,  2, 12,  0,  7,  5, 10,  6,  8,  9,  1]),
 array([ 8,  3,  6, 14,  7, 15,  2, 12,  4, 11,  1,  9,  0, 10, 13,  5]),
 array([11,  1, 10, 12,  3,  5,  0, 14, 15,  6,  4, 13,  7,  8,  2,  9]),
 array([ 8,  3,  6, 11,  7, 15,  2, 12,  4, 14,  1,  9,  0, 10, 13,  5]),
 array([11,  1, 10, 12,  3,  5,  0, 14, 15,  6,  4, 13,  7,  8,  2,  9]),
 array([ 6,  9, 13, 10, 11, 12,  3, 14,  7,  5,  4,  1, 15,  2,  8,  0]),
 array([ 6,  9, 13, 10, 11, 12,  3, 14,  7,  5,  4,  1, 15,  2,  8,  0]),
 array([11, 12,  8,  3,  4, 14,  7, 15,  0,  2,  1,  9, 10,  5, 13,  6]),
 array([14, 13,  1, 10,  5,  6, 12,  2,  4,  0, 11,  9,  3,  8, 15,  7]),
 array([ 9,  6, 14,  8,  7,  2, 15, 13, 12, 10,  1,  4,  3,  0,  5, 11])]
```

In [28]:
```
#inisialisasi jumlah kota
jumlah_kota = 16
#inisialisasi ukuran populasi awal
popSize = 10
#inisialisasi iterasi maksimum
iterasi_maksimum = 5000
#inisialisasi crossover rate (cr)
cr = 0.4
#inisialisasi mutation rate (mr)
mr = 0.1
# buat populasi awal
population = populasi(jumlah_kota, popSize)
#inisialisasi array kosong untuk menyimpan individu terbaik setiap generasi
individu_terbaik_generasi = []
#inisialisasi array kosong untuk menyimpan fitness individu terbaik setiap generas
fitness_terbaik_generasi = []
#lalkukan perulangan sebanyak jumlah iterasi maksimum
#gabung_array2 = np.concatenate((individu_terbaik_generasi,fitness_terbaik_generas

fitness_terbaik = []
individu_terbaik = []
for i in range(iterasi_maksimum):
    #print('hasil_populasi :',i)
    #for j in range(len(population)):
    #    print(population[j])
    #print('\n')
    hasil_cross = crossover(cr,population)
    hasil_mutasi = mutasi(mr,hasil_cross,popSize)
    hasil_fitness = fitness(hasil_mutasi)
    hasil_seleksi = seleksi_roulette(jumlah_kota,hasil_fitness,hasil_mutasi,popSi
    fitness_hasilSeleksi = fitness(hasil_seleksi)
    population = hasil_seleksi

    #untuk mendapatkan individu terbaik dari tiap generasi
    sort = []

    for j in range(len(hasil_seleksi)):
        sort.append(j)
    sort = np.reshape(sort,(len(sort),1))
    #menggabungkan index array populasi  dengan fitness hasil seleksi
    gabung_array = np.concatenate((sort,fitness_hasilSeleksi),axis = 1)
    #mengurutkan berdasarkan fitness tertinggi
    hasil_sort = gabung_array[gabung_array[:,1].argsort()][::-1]
    #print(hasil_sort)
    #int(hasil_sort[0])
    #menambahkan individu terbaik generasi ke dalam array
    individu_terbaik_generasi.append(hasil_seleksi[int(hasil_sort[0][0])])
    #menambahkan fitness terbaik generasi ke dalam array
    fitness_terbaik_generasi.append(hasil_sort[0][1])

    #untuk mendapatkan individu dan fitness terbaik keseluruhan
    if(all(i <= hasil_sort[0][1] for i in fitness_terbaik_generasi)):
        fitness_terbaik = []
        individu_terbaik = []
        individu_terbaik.append(hasil_seleksi[int(hasil_sort[0][0])])
        fitness_terbaik.append(hasil_sort[0][1])
```

```
#    print('hasil seleksi generasi :',i)
#    print(hasil_sort[0][1])
#    for j in range(len(hasil_seleksi)):
#        print(hasil_seleksi[j])
#    print('\n')


print('individu terbaik yang didapatkan adalah :',individu_terbaik,'dengan fitnes
#digunakan untuk plotting grafik
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(fitness_terbaik_generasi)
plt.xlabel('Generasi')
plt.ylabel('nilai fitness terbaik')
plt.show()
```

```
individu terbaik yang didapatkan adalah : [array([10,  5,  0,  6,  7,  1, 11,
4, 15, 14, 12,  8,  9,  3, 13,  2])] dengan fitness: [0.00033277870216306157]
```



Daftar Pustaka :

1.) Mahmudy, Wayan Firdaus; Dewi, Candra.(2014). Implementasi Algoritma Genetika pada Optimasi Biaya Pemenuhan Kebutuhan Gizi.

2.) Mahmudy, Wayan Firdaus.(2015). Dasar - dasar Algoritma Evolusi

3.) Andhyka, Awang.(2018). Penerapan Algoritma Genetika pada permasalahan Matematika

4.) Amelia, Lily; Aprianto.(2011). Optimalisasi penjadwalan produksi dengan metode algoritma genetika di PT.Progress Diecast

5.) Febriyana, Ria; Mahmudy, Wayan Firdaus.(2016). Penjadwalan kapal penyeberangan menggunakan algoritma Genetika

6.) Damayanti, Madi Putri, M.Ali Fauzi.(2017).implementasi algoritma genetika untuk penjadwalan Customer service

7.) Fox,Roland dan Steven Tseng.1997.Traveling Salesman Problem by Genetic Algorithm and Simulated Annealing.

8.) Kumar, Anit.(2013)Encoding Schemes in Genetic Algorithm