Laporan Tugas Besar 1 IF3170 Intelegensi Artifisial

Semester I Tahun Akademik 2024/2025

Penyelesaian Persoalan Pencarian Solusi Diagonal Magic Cube 5x5 dengan Local Search

Disusun Oleh

Edbert Eddyson Gunawan	13522039
Vanson Kurnialim	13522049
Muhammad Syarafi Akmal	13522076
Fabian Radenta Bangun	13522105

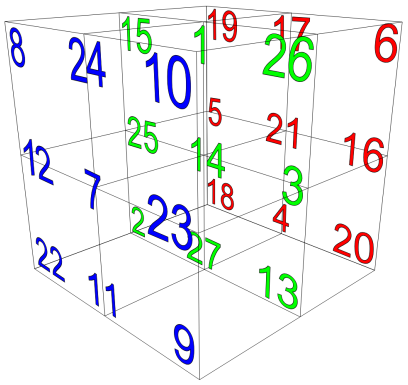
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung

2024

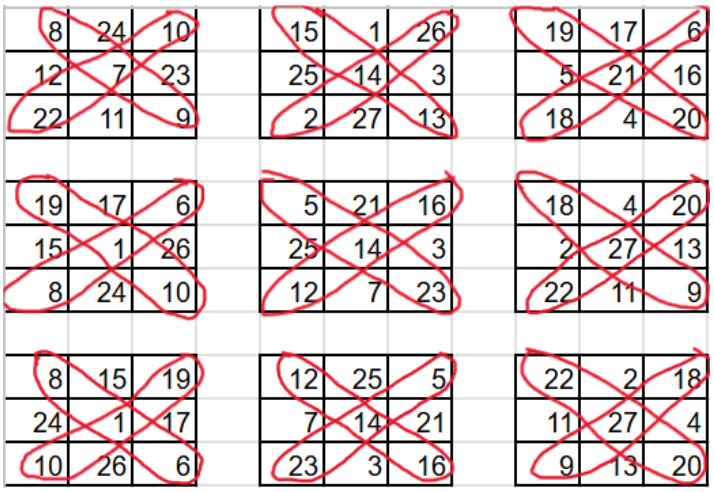
Deskripsi Persoalan

Diagonal magic cube merupakan kubus yang tersusun dari angka 1 hingga n^3 tanpa pengulangan dengan n adalah panjang sisi pada kubus tersebut. Angka-angka pada tersusun sedemikian rupa sehingga properti-properti berikut terpenuhi:

- Terdapat satu angka yang merupakan magic number dari kubus tersebut (Magic number tidak harus termasuk dalam rentang 1 hingga n^3, magic number juga bukan termasuk ke dalam angka yang harus dimasukkan ke dalam kubus)
- Jumlah angka-angka untuk setiap baris sama dengan magic number
- Jumlah angka-angka untuk setiap kolom sama dengan magic number
- Jumlah angka-angka untuk setiap tiang sama dengan magic number
- Jumlah angka-angka untuk seluruh diagonal ruang pada kubus sama dengan magic number
- Jumlah angka-angka untuk seluruh diagonal pada suatu potongan bidang dari kubus sama dengan magic number
    - Berikut ilustrasi dari potongan bidang yang ada pada suatu kubus berukuran 3:



    - Terdapat 9 potongan bidang, yaitu:



    - Diagonal yang dimaksud adalah yang dilingkari warna merah saja
- Ilustrasi dan penjelasan lebih detail bisa anda lihat di link berikut: Features of the magic cube - Magisch vierkant

Pada tugas ini, peserta kuliah akan menyelesaikan permasalahan Diagonal Magic Cube berukuran 5x5x5. Initial state dari suatu kubus adalah susunan angka 1 hingga 5^3 secara acak. Kemudian, tiap iterasi pada algoritma local search, langkah yang boleh dilakukan adalah menukar posisi dari 2 angka pada kubus tersebut (2 angka yang ditukar tidak harus bersebelahan). Khusus untuk genetic algorithm, boleh dilakukan penukaran posisi lebih dari 2 angka sekaligus dalam satu iterasi (tetap hanya menukar posisi 2 angka saja juga diperbolehkan).

Pembahasan

Pemilihan Fungsi Objektif

Fungsi objektif adalah nilai evaluasi yang digunakan untuk menilai setiap keadaan (state) pada pencarian lokal (local search), yang berperan sebagai parameter untuk menentukan keberhasilan. Fungsi objektif ini berkaitan erat dengan pencapaian global optimum dan local optimum.

Dalam permasalahan ini, **fungsi objektif yang kami pilih adalah menghitung jumlah komponen pada magic cube yang jumlahnya tidak sesuai dengan magic number**. Magic number adalah konstanta yang dihitung berdasarkan konfigurasi solusi dari magic cube. Nilai ini bersifat tetap karena magic cube memiliki constraint, yaitu setiap angka dalam *magic cube* merupakan enumerasi dari [1, n^3], di mana n adalah dimensi dari *magic cube*. Rumus magic number diberikan sebagai berikut:

$$\text{Magic Number} = n(n^3 + 1) \div 2$$

Magic Number = 315 (5x5x5 cube)

Objective function = sum of wrong magic cube components

Global Optimum = 0 (no wrong magic cube components)

Kami memilih fungsi objektif ini karena, menurut kami, bentuk state value ini memberikan metode evaluasi yang paling konkret untuk menilai setiap keadaan.

Algoritma Stochastic Hill Climbing

Algoritma Stochastic Hill Climbing adalah algoritma local search yang mencari solusi secara bertahap dengan mengambil sebuah successor random dari state saat ini, kemudian membandingkan antara nilai objective function dari state sekarang dan nilai objective function dari successor yang diambil. Jika nilai objective function dari successor tersebut lebih baik maka successor tersebut akan diambil sebagai state saat ini. Namun, jika nilai objective function dari successor tidak lebih baik maka successor tidak diambil dan pencarian dilakukan kembali dengan state yang sama.

Pada kasus permasalahan Magic Cube 5x5 ini, konfigurasi awal dari pencarian adalah kondisi sebuah kubus hasil random yang sudah terisi lengkap dengan angka 1 sampai 125 masing-masing muncul sekali tanpa perulangan. Kemudian langkah yang dilakukan untuk mengubah konfigurasi kubus menuju solusi pada setiap iterasinya adalah menukar posisi dari 2 angka pada kubus. Penukaran posisi pada kubus terus dilakukan hingga iterasi dilakukan sebanyak maksimum iterasi. Berikut adalah *source code* implementasi algoritma Stochastic Hill Climbing:

```
package algorithm

import (
	"fmt"
	"math/rand"
	"time"
)

const MAX_ITERATION = 100

var final_objective_value int
var objective_value_list[MAX_ITERATION]int

func StochasticHC() {
	var current_objective_value, neighbor_objective_value, start_x, start_y,
start_z, destination_x, destination_y, destination_z int
	var objective_value_list [MAX_ITERATION]int
	magic_cube := CreateCube() // [][][]int

	// saved initial state
	saved_magic_cube := CopyCube(magic_cube)
	var saved_steps []CoordinatePair
```

```
	fmt.Println("State awal kubus :")
	ShowMatrixXZ(magic_cube)

	start_time := time.Now()
	for i := 0; i < MAX_ITERATION; i++ {
		// fmt.Printf("Pengulangan ke-%d\n", i+1)
		current_objective_value = EvaluateObjectiveFunction(&magic_cube)

		// indeks pada current magic cube yang ingin di swap
		start_x = rand.Intn(MATRIX_N)
		start_y = rand.Intn(MATRIX_N)
		start_z = rand.Intn(MATRIX_N)

		// indeks tujuan swapping
		destination_x = rand.Intn(MATRIX_N)
		destination_y = rand.Intn(MATRIX_N)
		destination_z = rand.Intn(MATRIX_N)

		// dapetin neighbor dengan swap start dan destination
		Swap(&magic_cube, start_x, start_y, start_z, destination_x,
destination_y, destination_z)
		neighbor_objective_value = EvaluateObjectiveFunction(&magic_cube)

		var cek_batal bool = false
		if neighbor_objective_value > current_objective_value {
			// dibalikin ke awal
			cek_batal = true
			Swap(&magic_cube, start_x, start_y, start_z, destination_x,
destination_y, destination_z)
		}
	    final_objective_value =
EvaluateX(&magic_cube)+EvaluateY(&magic_cube)+EvaluateZ(&magic_cube)
	    objective_value_list[i] = final_objective_value
		if final_objective_value == 0 {
			fmt.Println("ketemu euy gacor!!")
			break
		}

		final_objective_value = EvaluateObjectiveFunction(&magic_cube)
		objective_value_list[i] = final_objective_value

		if !cek_batal {
			// SAVED DATA FOR VISUALIZATION
			var new_data CoordinatePair

			var kordinat1 Coordinate3D
			kordinat1.X = start_x
			kordinat1.Y = start_y
			kordinat1.Z = start_z
```

```
                    var kordinat2 Coordinate3D
                    kordinat2.X = destination_x
                    kordinat2.Y = destination_y
                    kordinat2.Z = destination_z

                    new_data.N = final_objective_value
                    new_data.Point1 = kordinat1
                    new_data.Point2 = kordinat2

                    saved_steps = append(saved_steps, new_data)
                }
            }

            duration := time.Since(start_time)

            SaveMatrixXZ(saved_magic_cube, saved_steps, "edbert.txt")
            // print
            fmt.Println("State akhir kubus :")
            ShowMatrixXZ(magic_cube)

            fmt.Println("Maksimum iterasi :", MAX_ITERATION)
            fmt.Println("Nilai fungsi objektif terakhir:", final_objective_value)
            fmt.Println("Waktu eksekusi:", duration)

}
```

Langkah kerja fungsi StochasticHC() pada source code diatas adalah sebagai berikut:

- Inisiasi konfigurasi awal (initial state) magic_cube dengan memanggil CreateCube().
- Menampilkan konfigurasi awal
- Melakukan iterasi mulai dari 0 hingga 1000000 (nilai maksimum iterasi yang diambil adalah 100000) dan pada tiap iterasi dilakukan:
  - Tentukan random successor sebagai neighbor dengan mengambil dua indeks random dengan setiap komponen x,y, dan z ditentukan dengan fungsi rand.Intn sebagai indeks dari dua buah angka yang akan ditukar
  - Lakukan penukaran pada kedua angka tersebut untuk mendapatkan neighbor
  - Jika nilai fungsi objektif neighbor tidak lebih baik (pada kasus ini lebih besar) dari nilai fungsi objektif current state, tukar kembali kedua indeks yang telah ditukar tadi (dikembalikan) sehingga konfigurasi cube kembali seperti semula
- Tampilkan informasi maksimum iterasi, nilai fungsi objektif current state terakhir, dan waktu eksekusi.

Eksperimen 1
Jumlah iterasi : 10000000

State awal kubus :
60 93 87 13 23
98 65 14 1 2
47 101 30 88 68
120 94 55 124 91
74 114 97 52 90

80 17 81 49 106
57 77 85 125 54
104 70 96 112 4
22 86 113 8 7
18 63 89 103 66

123 16 119 33 46
58 115 29 9 35
78 75 95 61 105
116 41 118 34 84
99 31 122 62 12

37 76 26 39 108
43 92 79 121 107
42 82 10 24 100
73 20 72 109 69
32 102 53 28 15

117 3 11 6 25
110 44 45 40 56
51 5 71 19 50
21 64 27 111 38
36 67 48 59 83

State akhir kubus :
38 86 56 42 93
5 102 97 1 113
79 67 94 39 36
125 65 4 88 33
68 111 64 32 40

66 13 107 48 81
108 63 41 87 16
98 23 49 30 115
37 92 112 54 20
6 121 18 29 83

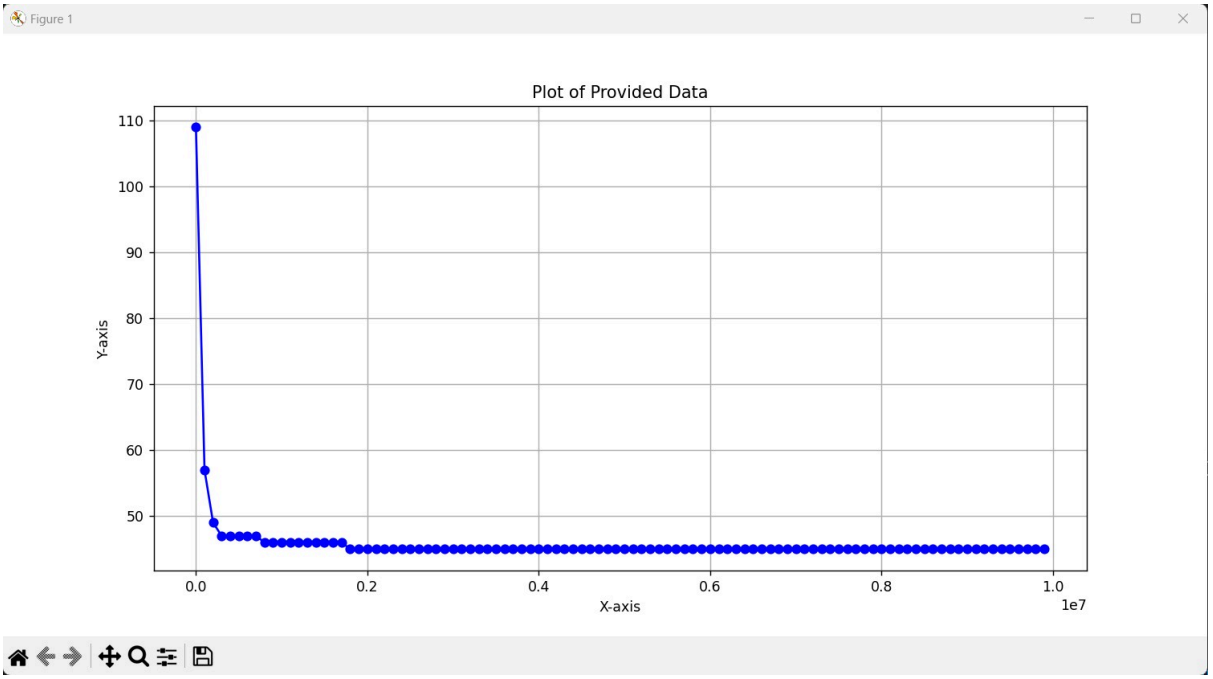
120 15 46 90 55
124 10 80 31 70
19 104 34 105 53
50 101 73 77 14
117 8 82 12 74

95 85 3 11 61
69 26 2 96 122
72 22 78 84 59
44 45 75 89 62
35 58 118 119 27

43 116 103 28 25
9 114 21 100 71
47 99 60 57 52
110 24 51 7 76
106 17 109 123 91

Nilai *objective function* akhir : 45

Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati :



Durasi pencarian : 27.5555308s

Eksperimen 2

Jumlah iterasi : 10000000

State awal kubus :
10 46 44 42 86
107 120 65 118 58
113 98 12 100 39
83 35 33 6 96
37 67 21 3 11

125 111 80 77 87
102 51 34 74 49
108 106 114 17 60
93 90 57 124 92
82 50 55 48 64

66 7 89 68 75
71 62 109 14 61
5 31 99 104 97
36 20 54 91 53
119 85 76 4 25

23 59 19 122 32
29 63 70 8 2
15 40 81 22 95
110 41 94 56 105
26 84 121 1 101

52 88 38 43 116
79 45 72 24 117
123 112 28 78 103
9 13 30 18 47
27 69 16 73 115

State akhir kubus :
19 115 75 37 92
39 13 106 52 117
103 27 61 14 89
97 53 43 124 71
57 107 30 88 33

94 42 114 7 64
118 65 36 85 104
32 1 26 5 50
22 91 123 35 2

49 47 16 108 95

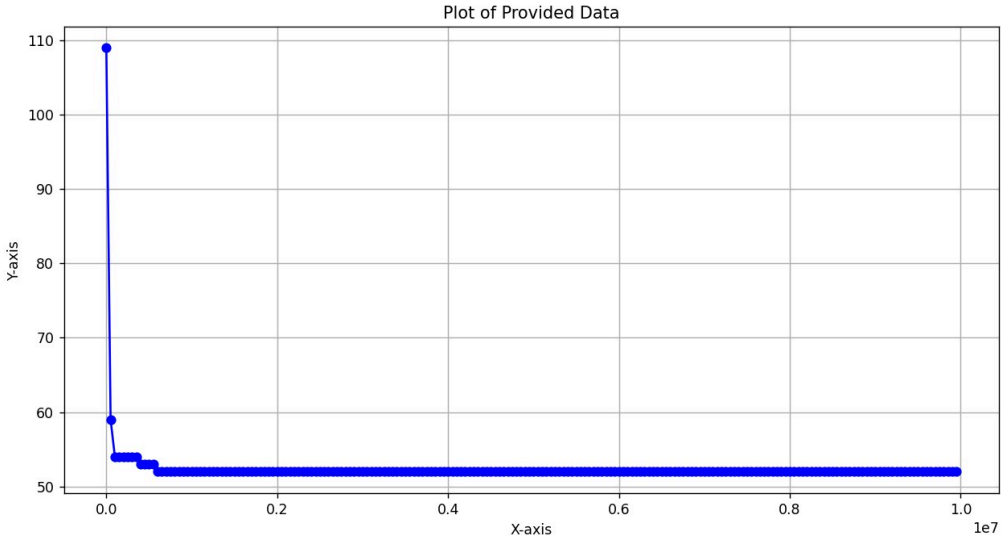
70 102 122 116 21
51 109 18 113 24
83 87 44 46 55
76 77 90 34 38
60 110 81 6 58

112 25 96 54 28
84 72 73 45 3
93 80 121 105 59
9 8 74 99 125
17 11 15 12 100

69 31 41 101 78
23 56 82 20 67
4 120 63 66 62
111 68 119 98 79
86 40 10 48 29

Nilai *objective function* akhir : 52

Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati :



Durasi pencarian : 28.3622585s

Eksperimen 3
Jumlah iterasi : 10000000

State awal kubus :
37 27 72 66 25
116 123 45 91 106
64 5 21 101 38
22 20 85 119 49
110 108 78 26 115

73 69 30 61 41
60 122 95 9 39
59 112 17 80 62
65 6 104 40 96
75 90 14 103 124

76 28 83 44 16
121 24 58 68 36
34 94 51 74 32
48 70 117 13 35
98 63 105 71 84

86 33 57 8 67
50 54 109 53 46
15 82 120 7 29
92 3 10 1 2
118 23 100 88 77

87 111 97 18 43
31 12 11 89 125
52 102 56 114 4
93 19 81 55 42
99 47 113 107 79

State akhir kubus :
118 10 26 100 61
1 12 102 18 4
70 14 66 48 117
6 116 92 43 58
54 51 29 106 75

83 21 35 93 96
15 122 125 112 94
89 76 31 17 7
65 77 56 39 78
63 19 95 98 40

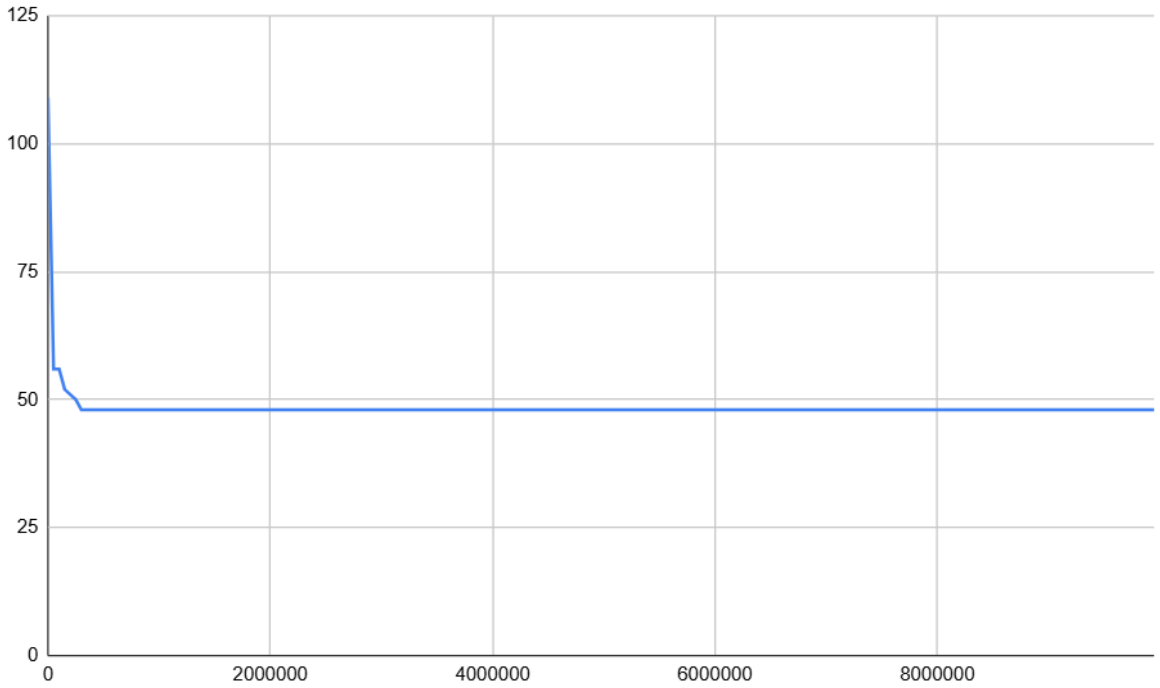
34 80 47 108 90
107 119 16 68 5
45 49 59 88 74
38 23 103 27 124
111 44 114 24 22

20 3 120 57 115
91 82 84 53 11
33 113 67 110 71
121 30 36 41 13
50 97 8 55 105

60 62 87 86 72
101 2 25 64 123
9 109 99 52 46
85 79 28 81 42
37 104 69 32 73

Nilai *objective function* akhir : 48

Plot nilai *objective function* terhadap banyak iterasi yang telah dilewati :



Durasi pencarian : 28.4250543s

Pembahasan

Berdasarkan eksperimen 1, 2, dan 3, nilai fungsi objektif akhir bervariasi antara 45 hingga 52. Hal ini menunjukkan bahwa StochasticHC mampu mencapai hasil yang cukup dekat dengan solusi optimal tetapi tidak selalu memberikan nilai terbaik yang konsisten di setiap eksperimen. Variasi ini bisa diakibatkan oleh komponen acak dalam pemilihan neighbor pada setiap iterasi.

Selain faktor acak, jumlah iterasi juga berpengaruh terhadap hasil. Jika jumlah iterasi ditambahkan, akurasi dari hasil pencarian akan ikut bertambah mendekati solusi. Hal ini tentu dapat terjadi karena semakin banyak jumlah iterasi, semakin banyak pula kemungkinan-kemungkinan jalan menuju penyelesaian yang dieksplorasi.

Algoritma Simulated Annealing

Algoritma Simulated Annealing adalah algoritma local search yang merupakan perpaduan antara Stochastic Hill Climbing dan pure random walk. Algoritma ini didasarkan pada proses pendinginan besi atau kaca (Annealing). Algoritma ini menggunakan simulasi penurunan suhu (T) sebagai parameter untuk keberjalanan algoritma pada dua komponen, yaitu terminasi pada saat suhu bernilai 0 dan parameter probabilistik yang digunakan untuk penentuan pemilihan langkah yang lebih buruk menggunakan rumus:

$$p = e^{\Delta E/T}$$

ΔE = perubahan energi (state value)

e = konstanta euler

T = suhu

Pada algoritma ini, saat ΔE bernilai negatif atau nol (pilihan lebih buruk), penerusan langkah ini akan ditentukan dari nilai probabilitas dari rumusan di atas dengan suatu *threshold* yang sudah ditentukan sebagai parameter batasan apabila dia kurang dari *threshold* maka akan langkah akan dilewati dan akan lanjut ke proses pencarian successor selanjutnya. Apabila ΔE bernilai positif maka (pilihan lebih bagus) maka dia sama saja seperti algoritma Stochastic Hill Climbing. Berikut adalah *source code* dari algoritma ini:

```
func MainSimulatedAnnealing() { // main interface for the algorithm
    T0 := math.Pow(10.0, 200.0)
    t := 1
    local_stuck := 0
    var p_plot []float64
    magic_cube := CreateCube()

    fmt.Println("State awal kubus: ")
    ShowMatrixXZ(magic_cube)

    start_time := time.Now()
    for t < 10 { // minimum value represented in float64
        TemperatureDecrease(&T0)

        current_objective_value := EvaluateObjectiveFunction(&magic_cube)
```

```
        // indeks pada current magic cube yang ingin di swap
        start_x := rand.Intn(MATRIX_N)
        start_y := rand.Intn(MATRIX_N)
        start_z := rand.Intn(MATRIX_N)

        // indeks tujuan swapping
        destination_x := rand.Intn(MATRIX_N)
        destination_y := rand.Intn(MATRIX_N)
        destination_z := rand.Intn(MATRIX_N)

        // dapetin neighbor dengan swap start dan destination
                Swap(&magic_cube,  start_x,  start_y,  start_z,  destination_x,
destination_y, destination_z)
        neighbor_objective_value := EvaluateObjectiveFunction(&magic_cube)

        if neighbor_objective_value == 0 {
            fmt.Println("Ketemu cuy")
            break
        }

        // kalkulasi delta_E
        delta_E := current_objective_value - neighbor_objective_value

        if delta_E <= 0 {
            p := Probabilistic(delta_E, T0)
            p_plot = append(p_plot, p)
            if p < 0.3 {
                // kondisi tidak diambil
                Swap(&magic_cube, destination_x, destination_y, destination_z,
start_x, start_y, start_z)
            }
            local_stuck++
        }

        // debugging purposes
        if(t % 1000 == 0){
            fmt.Println("T :", T0)
            fmt.Println("p: ", Probabilistic(delta_E, T0))
            fmt.Println("iteration:", t)
            fmt.Println("Objective Value:", current_objective_value)
            fmt.Println("")
        }
```

```
        t++
    }

    ShowMatrixXZ(magic_cube)
    duration := time.Since(start_time).Minutes()
    fmt.Println("time taken:", duration, "minute(s)")
    fmt.Println("iterations:", t)
    fmt.Println("Local optima frequency:", local_stuck)
}
```

```
func TemperatureDecrease(T *float64) {
    if *T > 1 {
        *T = *T * 0.99
    } else {
        *T = *T - (5 * math.Pow(10.0, -7.0))
    }
}
func Probabilistic(delta_E int, T float64) float64 {
    euler := 2.71828
    power := float64(float64(delta_E) / T)
    return math.Pow(euler, power)
}
```

Kode di atas memiliki 3 fungsi utama, yang 2-nya adalah fungsi *utilities* yang membantu keberjalanan fungsi utama. Berikut adalah langkah kerja dari fungsi Simulated Annealing di atas:

1. Proses inisiasi nilai T, state awal kubus yang random pada invokasi metode CreateCube(), dan variabel-variabel lainnya.
2. Proses looping dengan kondisi dimana selama T > 5 x 10 ^ -323 (nilai representasi minimum pada tipe float64) maka loop akan terus berjalan.
3. Pada awal loop, dilakukan proses penurunan temperatur dengan invokasi metode TemperatureDecrease(&T0) yang menerima parameter *reference* (pointer) dari suhu.
4. Proses sisa pada loop berupa proses kondisional yang melihat nilai ΔE (yang didapat dengan: current_objective_value - neighbor_objective_value, ini berbalik dengan salindia karena nilai objektifnya semakin kecil semakin bagus) , apabila bernilai lebih dari nol maka akan langsung dipilih menjadi next successor, namun apabila negatif maka akan masuk ke percabangan probabilistik dengan nilai p yang diinvokasi oleh metode Probabilistic(T, delta_E).

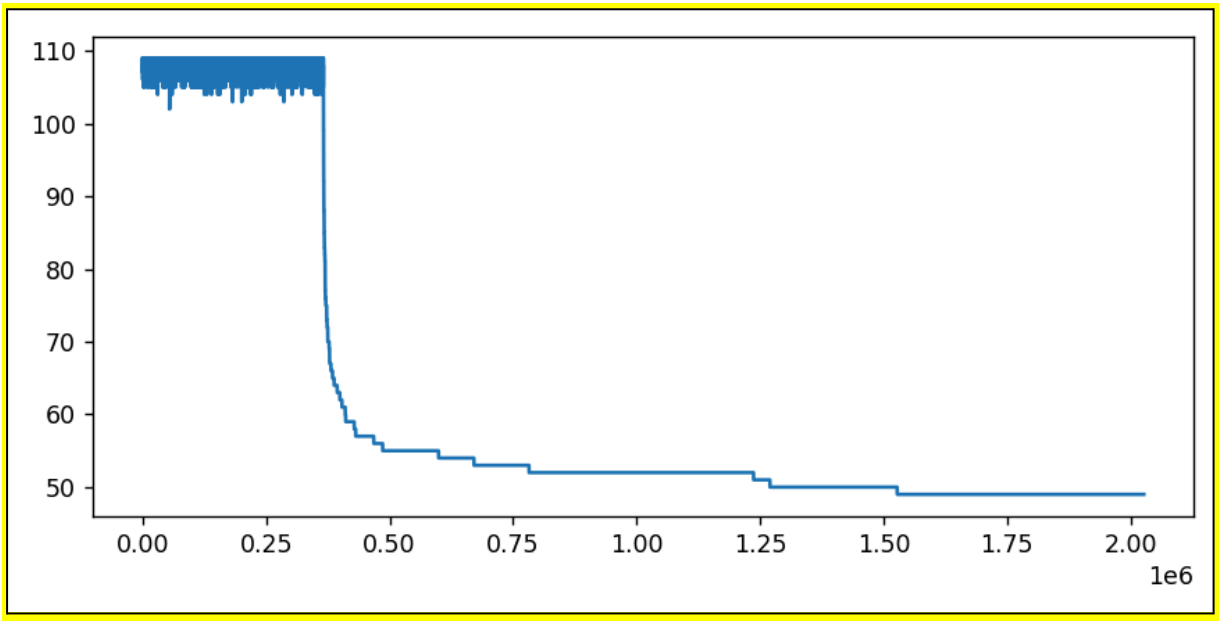
5. Proses ini berlanjut hingga ditemukan solusi (objective function bernilai nol), atau sudah masuk proses terminasi.

Eksperimen 1

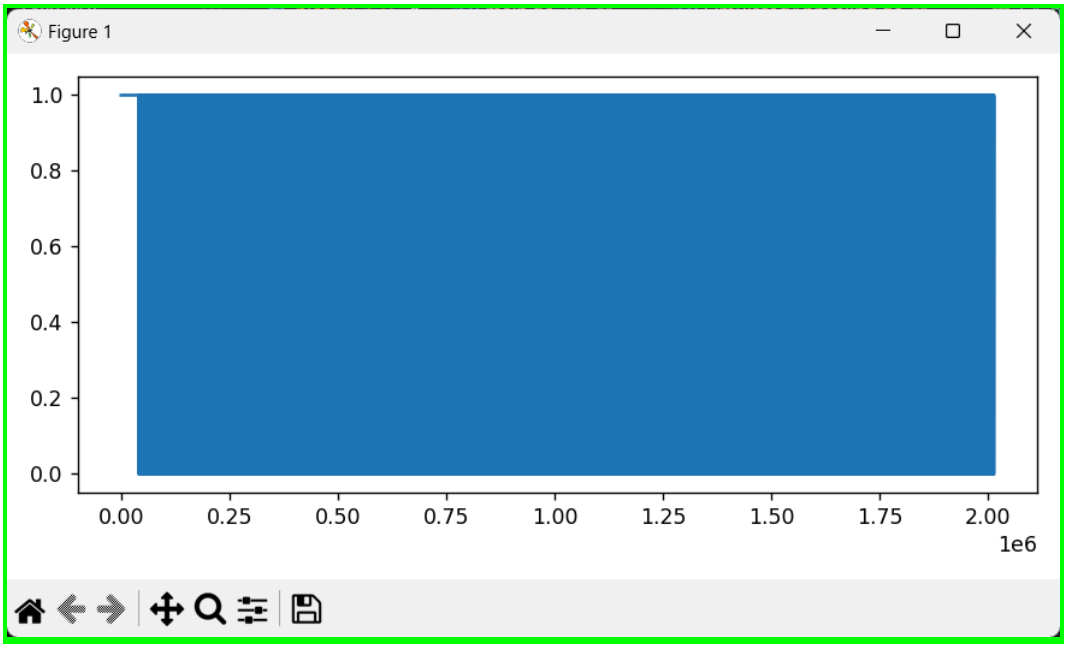
```
time taken: 1.9366246 s
Objective Value: 49
iterations: 2026970
Local optima frequency: 2012977
```

```
State awal kubus:
71 1 81 120 19
49 72 105 114 2
21 7 30 61 55
27 67 110 86 64
23 65 118 123 54

14 106 113 70 119
33 44 22 82 99
38 116 94 29 37
32 31 46 96 69
109 51 75 36 78

107 121 102 63 45
48 115 56 85 95
111 91 43 108 98
89 125 60 53 88
66 79 40 25 42

68 74 100 12 83
17 77 41 35 104
10 13 6 15 28
52 57 58 8 73
84 20 47 16 59

62 4 97 39 124
93 11 92 87 24
18 5 80 90 9
34 50 122 76 101
117 103 112 26 3
```

```
State akhir kubus:
85 33 108 38 51
83 76 7 81 68
49 122 18 109 17
39 92 115 13 56
53 66 67 6 123

86 46 117 50 16
55 2 5 47 120
59 74 63 26 93
8 82 100 104 21
107 57 30 88 65

43 73 77 103 69
114 25 15 31 61
9 14 89 24 79
95 90 20 106 4
111 113 29 1 23

10 101 78 27 99
75 91 19 84 12
102 32 42 119 105
125 87 58 64 124
3 34 118 116 44

36 62 40 97 80
94 121 112 72 54
96 35 70 37 11
48 52 22 28 110
41 45 71 98 60
```



Plot fungsi objektif



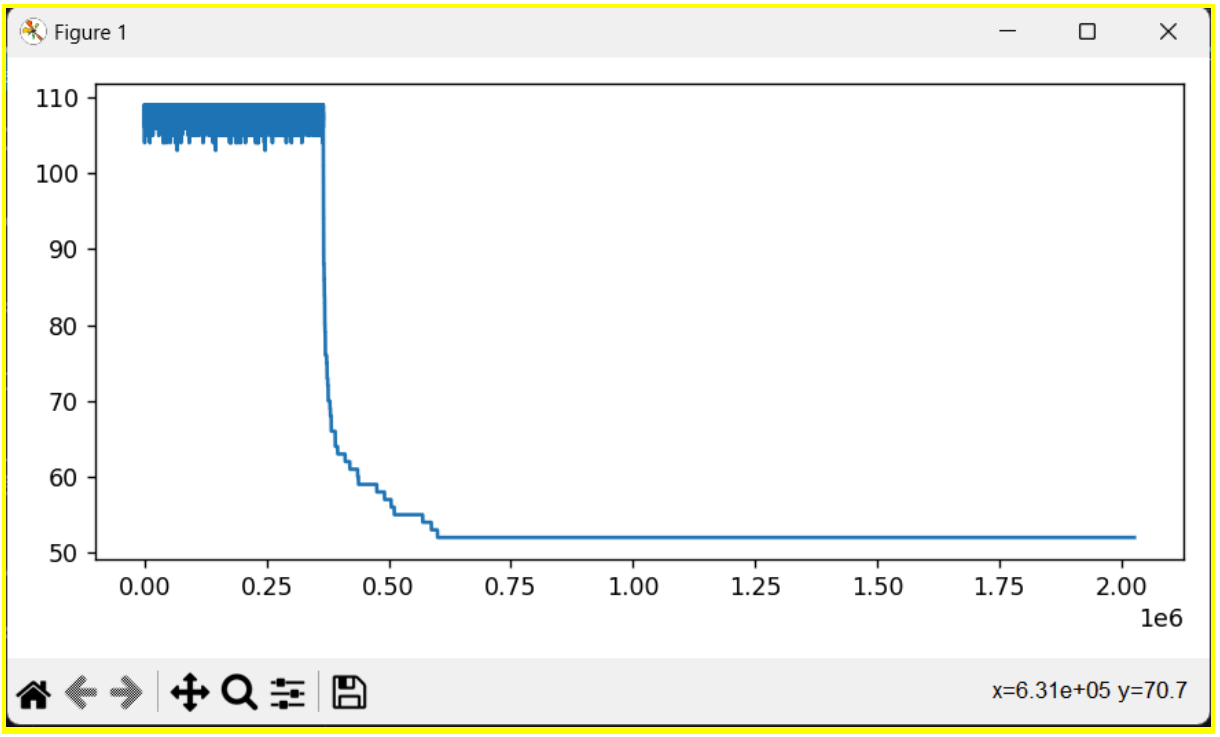
Plot p value

Eksperimen 2

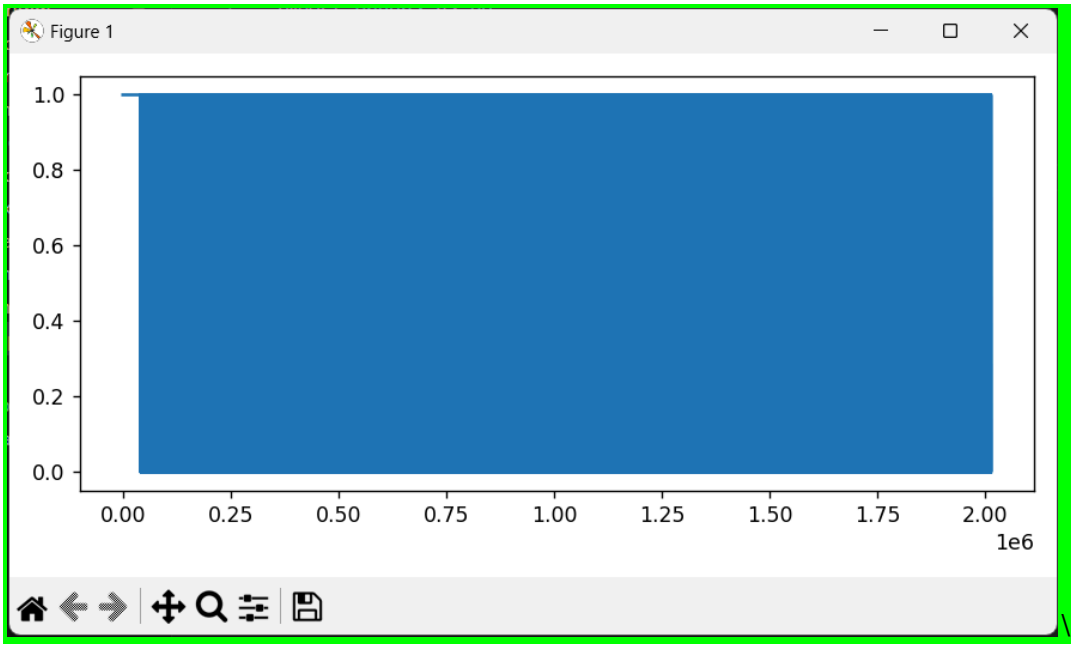
```
time taken: 1.9683705 s
Objective Value: 52
iterations: 2026970
Local optima frequency: 2012937
```

```
State awal kubus:
22 48 70 84 97
52 13 59 64 109
41 79 92 75 119
56 65 106 20 37
17 18 27 46 63

107 83 104 4 35
11 88 47 91 113
80 21 124 73 66
30 95 42 6 40
54 25 93 8 45

76 10 23 90 121
15 39 112 123 108
24 69 29 78 61
120 62 100 86 1
71 49 53 81 19

34 74 50 117 3
60 51 77 102 99
85 101 94 87 68
72 67 58 28 122
98 82 43 103 55

5 2 115 118 114
96 57 89 38 26
33 31 116 14 16
110 44 32 105 9
125 111 36 12 7
```

```
State akhir kubus:
16 106 59 5 107
69 92 8 89 57
102 23 81 33 76
4 48 37 79 9
121 20 41 49 1
111 103 119 55 114
17 82 105 7 104

113 99 115 96 53
85 21 101 18 90
36 95 86 52 124
43 120 73 31 2
38 25 88 118 46

109 12 122 14 58
100 44 78 61 32
19 97 112 110 84
42 39 77 63 94
45 123 29 67 47

15 70 6 75 10
34 93 66 68 54
60 80 74 71 30
108 64 24 50 65
98 27 56 51 83
```



Plot fungsi objektif



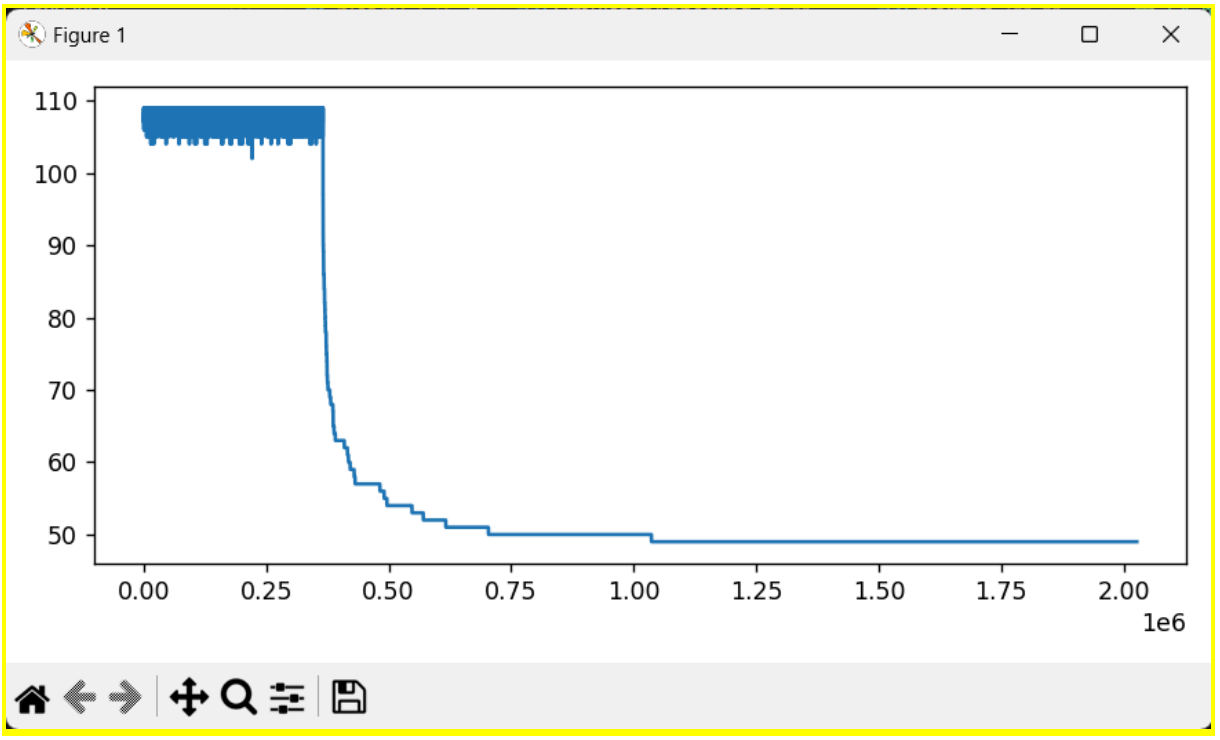
Plot p value

Eksperimen 3

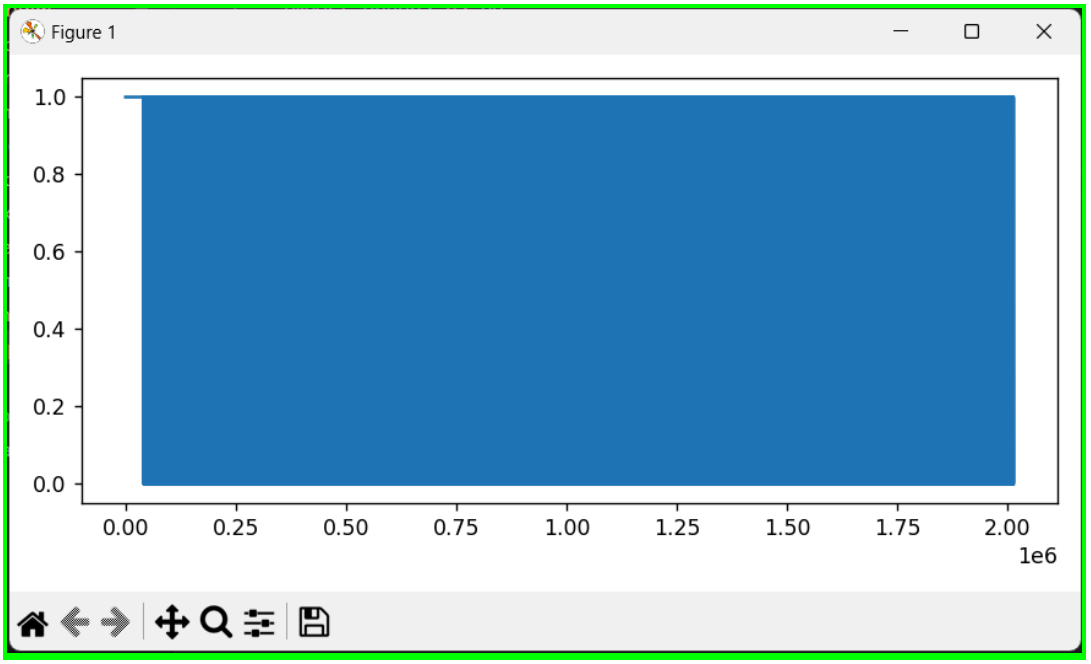
```
time taken: 1.8335004000000001 s
Objective Value: 53
iterations: 2026970
Local optima frequency: 2012989
```

```
State awal kubus:
106 39 29 103 88
15 31 4 66 23
21 6 14 75 67
119 107 104 71 125
72 18 58 114 99

61 37 112 76 92
50 10 117 102 40
109 34 19 3 111
47 65 12 32 84
110 38 62 17 74

43 53 46 87 95
90 100 91 16 70
35 68 73 11 108
51 41 82 79 63
27 30 48 33 26

113 8 93 115 69
59 22 120 116 13
2 57 122 20 1
83 123 45 98 55
78 36 118 44 64

121 96 49 86 54
94 77 81 56 5
42 85 89 105 60
101 7 80 97 24
52 124 25 9 28
```

```
State akhir kubus:
52 36 101 62 64
41 118 55 28 73
71 33 10 103 98
66 40 100 59 50
85 88 49 63 30

53 61 68 37 96
106 43 76 83 7
87 108 80 22 74
21 8 67 110 109
48 95 24 119 29

78 81 107 44 5
124 56 12 120 3
123 54 69 65 31
114 117 11 35 38
4 17 116 51 77

57 15 13 125 105
58 75 121 1 60
20 112 70 94 19
25 111 27 79 34
90 2 84 16 97

92 122 9 47 45
72 23 115 99 6
14 18 86 104 93
91 39 102 32 89
46 113 42 26 82
```



Plot fungsi objektif



Plot p value

Penjelasan

Berdasarkan tren pada ketiga eksperimen, berikut statistik rata-rata:

- Durasi: 1,85s

- Nilai objektif: 51,3

Apabila melihat dari ketiga plot nilai objektif, di awal terdapat seperti nilai yang naik turun, hal ini merupakan proses *random walk* dari algoritma Simulated Annealing, lalu setelah T menurun algoritma ini berubah menjadi stochastic.

Algoritma Genetik

Algoritma Genetik ini bekerja dengan saling menukar bagian random (*crossover*) pada *parent* untuk membentuk *child* baru. Berbeda dengan algoritma sebelumnya, fungsi objektif yang digunakan adalah *Fitness Value* atau banyak komponen pada *cube* yang sesuai dengan *Magic Number*. Hal ini dilakukan karena nilai tersebut digunakan untuk mengukur *value* keseluruhan dari sebuah *cube*. *Fitness Value* ini juga digunakan pada komposisi *roulette wheel selection* yang memberikan peluang lebih pada *cube* ber-*value* tinggi untuk dijadikan *parent*. Terdapat pula proses mutasi yang pada kasus ini dilakukan pada semua *child*. Mutasi dilakukan dengan menukar satu pasang titik random pada *cube*. Mutasi hanya dilakukan sekali untuk tiap *child* karena setelah pengujian singkat, hasil pencarian dengan mutasi banyak (20 untuk tiap *child*) menghasilkan hasil yang lebih buruk dibanding sekali mutasi, dinilai dari *Fitness Value* yang didapatkan.

Berikut adalah *source code* yang digunakan beserta penjelasannya :

```
func GeneticAlgorithm(populasi int, iterasi int) {
    start := time.Now()

    highestFitness := 0
    var bestCube [][][]int
    var stateAwal [][][]int
    fitnessArrayAvg := make([]float64, iterasi)
    fitnessArrayBest := make([]int, iterasi)
    kubusMagik := false
    indeksKubusMagik := -1

    ganjil := populasi%2 == 1

    cubesArray := make([][][][]int, populasi)
    for i := 0; i < populasi; i++ {
        cubesArray[i] = CreateCube()
        if i == 0 {
            stateAwal = cubesArray[i]
        }
    }

    for i := 0; i < iterasi; i++ {
        println("Iterasi ", i+1)
        if kubusMagik {
```

```
            break
        }
        // Ngisi fitness value untuk tiap cube
        totalFitness := 0
        fitnessValue := make([]int, populasi)
        fitnessAvg := 0.0
        fitnessLocalBest := 0
        for temp := 0; temp < populasi; temp++ {
                                            fitnessValue[temp]    =    109    -
EvaluateObjectiveFunction(&cubesArray[temp])
            fitnessAvg += float64(fitnessValue[temp])
            // cek fitness value terbagus global
            if fitnessValue[temp] > highestFitness {
                highestFitness = fitnessValue[temp]
                bestCube = cubesArray[temp]
            }
            // cek fitness value terbagus lokal
            if fitnessValue[temp] > fitnessLocalBest {
                fitnessLocalBest = fitnessValue[temp]
            }
            if fitnessValue[temp] == MAGIC_VALUE {
                kubusMagik = true
                indeksKubusMagik = temp
                break
            }
            totalFitness += fitnessValue[temp]
        }

        fitnessArrayBest[i] = fitnessLocalBest
        fitnessAvg = fitnessAvg / float64(populasi)
        fitnessArrayAvg[i] = fitnessAvg

        //bikin roulette wheel selection
        roulette := make([]float64, populasi)
        totalPersen := 0.0
        for temp := 0; temp < populasi; temp++ {
            bagi := float64(fitnessValue[temp]) / float64(totalFitness)
            totalPersen += float64(bagi)
            roulette[temp] = totalPersen
        }

        // menentukan parent
```

```
        parentCubes := make([][][][]int, populasi)
        for temp := 0; temp < populasi; temp++ {
            r := rand.Float64()
            for temp2 := 0; temp2 < populasi; temp2++ {
                if r <= float64(roulette[temp2]) {
                    parentCubes[temp] = cubesArray[temp2]
                    break
                }
            }
        }

        // PMX Crossover
        for j := 0; j < populasi; j += 2 {
            if j+1 < populasi {
                // Membuat 3D array jadi 1D
                parent1 := StraightCube(parentCubes[j])

                var parent2 []int
                    if ganjil && j == populasi-1 { // Kalo ganjil, populasi
terakhir nge-parent sama elemen sebelumnya (kedua terakhir)
                    parent2 = StraightCube(parentCubes[j-1])

                } else {
                    parent2 = StraightCube(parentCubes[j+1])
                }

                child1, child2 := pmxCrossover(parent1, parent2)

                SwapStraightRandom(&child1)
                SwapStraightRandom(&child2)

                // balikin jadi cube kotak
                cubesArray[j] = CubedCube(child1)
                if !ganjil { // jika ganjil ga ush tambahin anak ke-2
                    cubesArray[j+1] = CubedCube(child2)
                }
            }
        }

    }
    if kubusMagik {
        print("ditemukan kubus magik\n")
```

```
        print(indeksKubusMagik)
        fmt.Println(bestCube)
    } else {
        print("tidak ditemukan kubus magik\n")
        fmt.Println("STATE AWAL")
        fmt.Println(stateAwal)
        fmt.Println("BEST CUBE")
        fmt.Println(bestCube)
    }
    fmt.Println("Highest Fitness : ", highestFitness)
    duration := time.Since(start)
    fmt.Println("Time Taken : ", duration)
    fmt.Println("Number of Iteration : ", iterasi)
    fmt.Println("Array of maximum fitness per iteration : ")
    fmt.Println(fitnessArrayBest)
    fmt.Println("Array of average fitness per iteration : ")
    fmt.Println(fitnessArrayAvg)

}
```

Fungsi utama pada algoritma genetik yang diimplementasikan. Fungsi ini mengatur keseluruhan alur algoritma dan menyimpan data-data penting lainnya. Alur fungsi terjadi sebagai berikut :

- Inisialisasi variabel-variabel dan mengisi *random state* awal untuk tiap *cube* sebanyak populasi.
- Masuk proses *loop* sebanyak iterasi.
- Menghitung nilai *fitness* dan membentuk *roulette wheel selection*.
- Membentuk *parent* berdasarkan secara *random* berdasarkan *roulette wheel selection*.
- Melakukan *crossover* untuk secara berpasangan. Jika populasi ganjil, *parent* terakhir berpasangan dengan elemen kedua terakhir namun hanya satu *child* yang dibentuk agar jumlah populasi tetap terjaga.
- Dilakukan mutasi untuk setiap *child* dan dijadikan sebagai *parent.*
- Proses diulang hingga iterasi selesai atau ditemukan *goal state*.

```
func pmxCrossover(parent1, parent2 []int) ([]int, []int) {
    length := len(parent1)
    child1 := make([]int, length)
    child2 := make([]int, length)
```

```
    // inisiasi array kosong
    for i := 0; i < length; i++ {
        child1[i] = -1
        child2[i] = -1
    }

    // Randomly milih 2 titik buat rentang crossover, diset point1 lebih kecil
    point1 := rand.Intn(length)
    point2 := rand.Intn(length)
    if point1 > point2 {
        point1, point2 = point2, point1
    }

    // Masang rentang crossover ke child masing2
    for i := point1; i <= point2; i++ {
        child1[i] = parent2[i]
        child2[i] = parent1[i]
    }

    // ngisi sisa elemen di child yang masih kosong
    for i := 0; i < length; i++ {
        if i >= point1 && i <= point2 {
            continue
        }
        // println(i)
        fillValue(child1, i, parent1)
        fillValue(child2, i, parent2)
    }

    return child1, child2
}
```

Fungsi yang digunakan untuk melakukan *crossover* pada *parent* untuk menghasilkan *child*. Teknik *crossover* yang digunakan adalah *Partially Mapped Crossover* (PMX) yang memasangkan bagian *random* dari *parent* pada *child* yang berseberangan lalu mengisi bagian kosong pada *child* oleh elemen *parent* sejajar yang belum ada pada *child* tersebut sehingga batasan unik pada *cube child* tetap terjaga.

```
func fillValue(child []int, idx int, parent []int) {
```

```
    for _, val := range parent {
        if !contains(child, val) {
            child[idx] = val
            break
        }
    }
}

func contains(slice []int, value int) bool {
    for _, v := range slice {
        if v == value {
            return true
        }
    }
    return false
}

func StraightCube(cube [][][]int) []int {
    straightedCube := make([]int, 125)
    index := 0
    for x := 0; x < 5; x++ {
        for y := 0; y < 5; y++ {
            for z := 0; z < 5; z++ {
                straightedCube[index] = cube[x][y][z]
                index++
            }
        }
    }
    return straightedCube
}

func CubedCube(straightCube []int) [][][]int {
    cubedCube := make([][][]int, 5)
    for x := 0; x < 5; x++ {
        cubedCube[x] = make([][]int, 5)
        for y := 0; y < 5; y++ {
            cubedCube[x][y] = make([]int, 5)
        }
    }
    index := 0
    for x := 0; x < 5; x++ {
        for y := 0; y < 5; y++ {
```

```
            for z := 0; z < 5; z++ {
                cubedCube[x][y][z] = straightCube[index]
                index++
            }
        }
    }
    return cubedCube
}
```

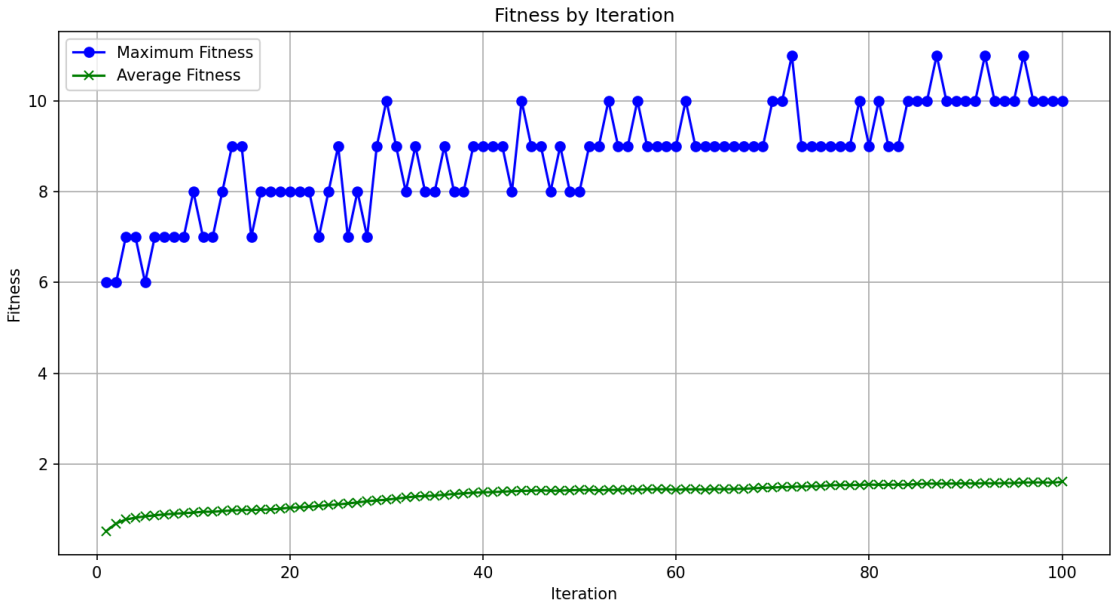
Fungsi fillValue untuk mengisi elemen kosong pada *child* sambil memastikan elemen yang diisi belum ada pada *child* tersebut. Fungsi contains mengembalikan boolean jika elemen masukan berada pada matrix. Fungsi StraightCube berfungsi untuk mengubah array 3 dimensi yang memuat *cube* menjadi array 1 dimensi. Kebalikannya, fungsi CubedCube mengubah array 1 dimensi kembali menjadi array 3 dimensi.

Hasil Experiment :

1. Iterasi = 100

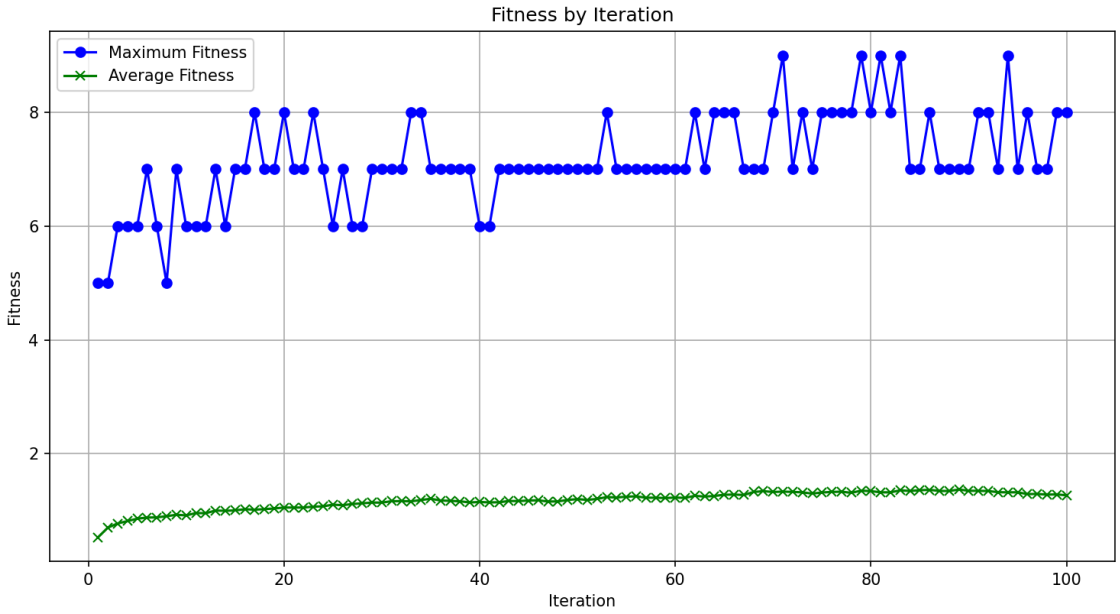
   1.1 Populasi 100.000

```
Highest Fitness :  11
Time Taken :  35m59.9307999s
Number of Iteration :  100
```



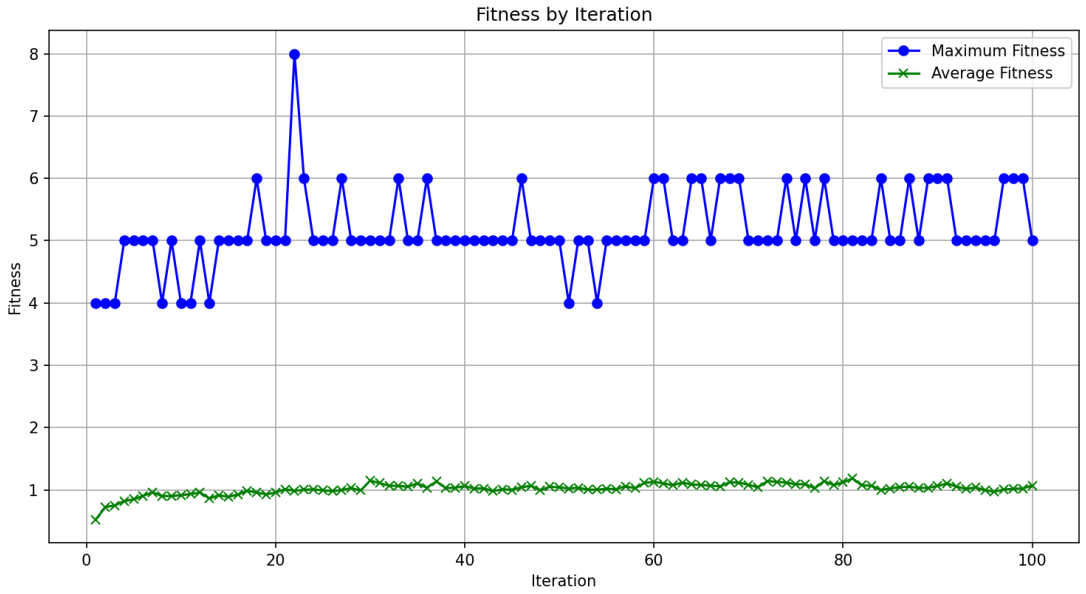
1.2. Populasi 10.000

```
Highest Fitness :  9
Time Taken :  3m0.0716114s
Number of Iteration :  100
Number of Population :  10000
```



1.3. Populasi 1000

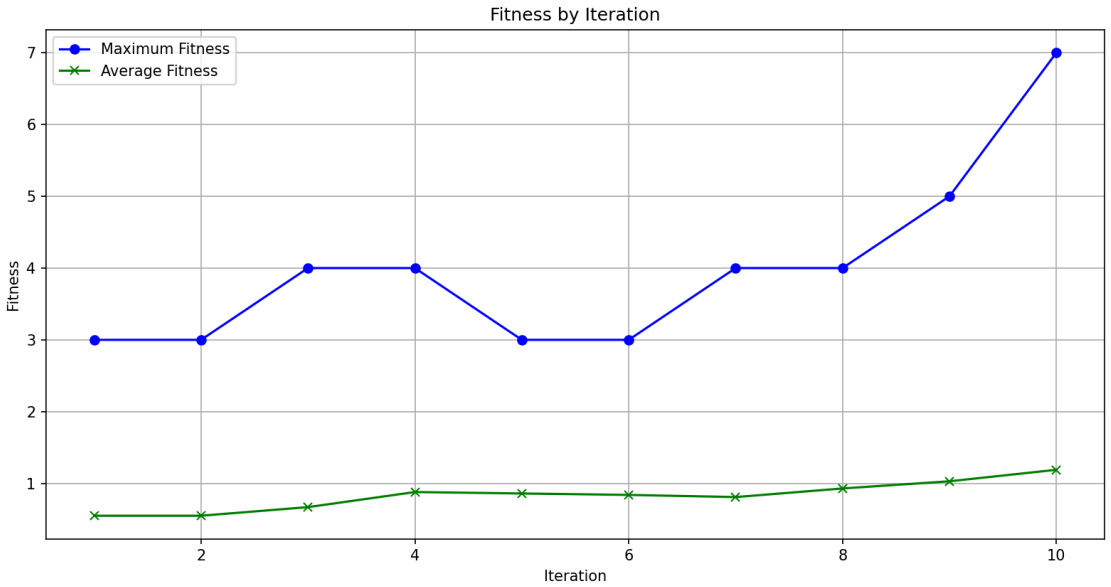
```
Highest Fitness :  8
Time Taken :  14.5222603s
Number of Iteration :  100
Number of Population :  1000
```



2. Populasi = 100

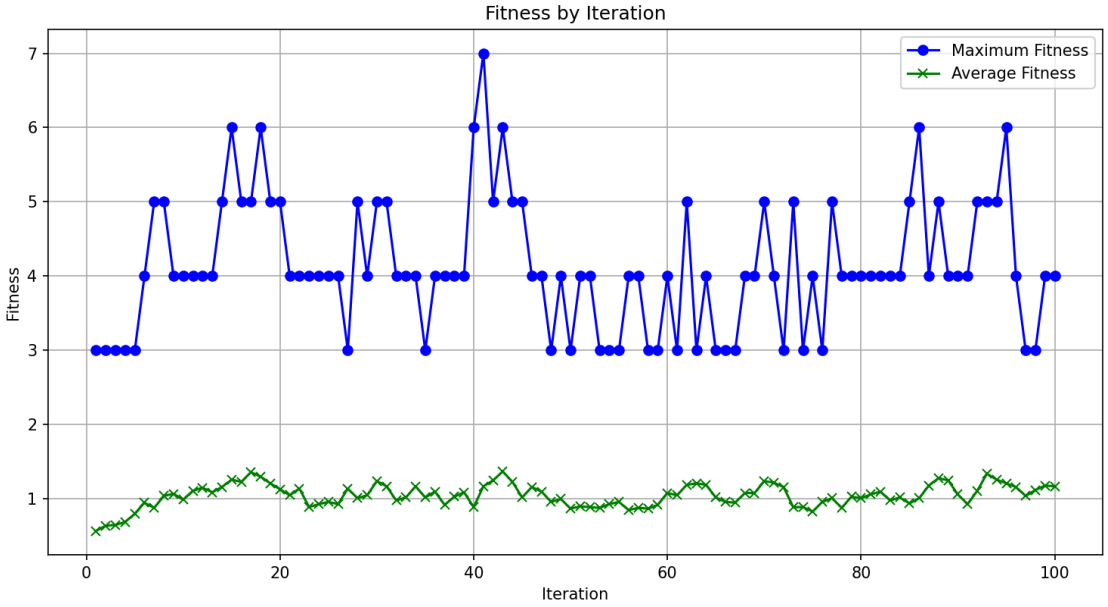
2.1 iterasi = 10

```
Highest Fitness :  7
Time Taken :  108.2275ms
Number of Iteration :  10
Number of Population :  100
```



2.2 iterasi = 100

```
Highest Fitness :  7
Time Taken :  1.6133753s
Number of Iteration :  100
Number of Population :  100
```



2.3 iterasi = 250

```
Highest Fitness :  7
Time Taken :  2.7429716s
Number of Iteration :  250
Number of Population :  100
```

Pada algoritma genetik, semakin banyak populasinya maka semakin baik hasil yang didapatkan, terlihat pada highest fitness pada data di atas. Sedangkan banyaknya iterasi tidak begitu memengaruhi efektifitas program. Hal tersebut mungkin terjadi karena iterasi yang digunakan kurang banyak / extreme.

Hasil yang didapatkan dari algoritma ini bahkan tidak mendekati *goal state*. Hal ini terjadi karena algoritma genetik melakukan *crossover* bagian secara random sehingga komponen magic pada cube yang sudah benar akan teracak kembali. Algoritma ini tidak memiliki kemampuan untuk menjaga fitness value yang dipunyai.

Dibandingkan dengan algoritma lain, Algoritma Genetik ini merupakan algoritma yang paling buruk karena konsep proses pencarian yang memang tidak cocok dengan kasus magic cube ini.

Kesimpulan

Berdasarkan hasil implementasi algoritma *local search* untuk menyelesaikan permasalahan *Diagonal Magic Cube* 5x5x5, dapat disimpulkan beberapa hal berikut:

1. **Stochastic Hill Climbing (SHC)** efektif untuk melakukan eksplorasi solusi dalam ruang keadaan yang besar dengan melakukan pertukaran posisi angka secara acak. Namun, karena sifatnya yang hanya bergerak ke tetangga yang sama dengan atau lebih baik, SHC cenderung mudah terjebak dalam local optimum dan memerlukan jumlah iterasi yang sangat besar untuk mencapai solusi optimal.
2. **Simulated Annealing (SA)** memberikan performa yang paling baik karena memungkinkan untuk random walk diawal yang kaan menghindari dari terjebak dalam local optima, tidak seperti stochastic dan genetic algorithm. Selanjutnya ketika sudah mendekati temperaturenya
3. **Genetic Algorithm (GA)** adalah menunjukkan perubahan yang b

Secara keseluruhan, dari ketiga algoritma local search yang diuji, Simulated Annealing menunjukkan hasil yang lebih baik dalam mencapai solusi mendekati optimal. SHC memiliki keterbatasan dalam pencarian solusi karena kecenderungannya untuk terjebak dalam local optimum, sedangkan GA tidak dapat menunjukkan hasil yang tepat.

Saran

Terdapat beberapa saran untuk melanjutkan pekerjaan tersebut.

1. Eksplorasi Parameter Tuning: untuk memperbaiki performa SA dan GA, tuning parameter seperti suhu awal, laju penurunan suhu pada SA, serta tingkat crossover dan mutasi pada GA dapat lebih dieksplorasi. Hal ini bisa dilakukan dengan metode grid search atau algoritma optimasi metaheuristik lainnya seperti Bayesian Optimization.
2. Menggunakan Parallel Computing / Concurrency: Mengingat kompleksitas perhitungan untuk ukuran masalah 5x5x5, penerapan parallel computing pada iterasi atau populasi yang besar akan mengurangi waktu komputasi dan memungkinkan eksplorasi ruang solusi yang lebih luas dalam waktu yang lebih singkat.

Pembagian Tugas

Task	**NIM**
Stochastic Hill Climbing Algorithm	13522105
Simulated Annealing Algorithm	13522076
Genetic Algorithm	13522049
Cube Class	13522039
Visualization [Bonus]	13522039
Dokumen Laporan	13522039, 13522049, 13522076, 13522105

Referensi

Institut Teknologi Bandung. (n.d.). *Hill climbing search* [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/194228-Beyond-Classical-Search/1693804849395_IF3170_Materi3_Seg03_BeyondClassicalSearch_HillClimbing.pdf

Institut Teknologi Bandung. (n.d.). *Simulated annealing* [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/53145-Artificial-Intelligence-Parallel-Class/194228-Beyond-Classical-Search/1693804872404_IF3170_Materi03_Seg04_BeyondClassicalSearch_SimulatedAnnealing.pdf

Institut Teknologi Bandung. (n.d.). *Genetic algorithm* [PDF]. Retrieved from https://cdn-edunex.itb.ac.id/storages/files/1727405202098_IF3170_Materi03_Seg05_BeyondClassicalSearch.pdf

Magisch Vierkant. (n.d.). *Features of the magic cube*. Retrieved from https://www.magischvierkant.com/three-dimensional-eng/magic-features/

Trump, R. (n.d.). *Perfect magic cubes*. Retrieved from https://www.trump.de/magic-squares/magic-cubes/cubes-1.html

Wikipedia contributors. (2023, October 20). *Magic cube*. In *Wikipedia, The Free Encyclopedia*. Retrieved from https://en.wikipedia.org/wiki/Magic_cube

National University of Singapore. (n.d.). *Annealing schedule*. Retrieved from https://phyweb.physics.nus.edu.sg/~phywjs/CZ3205/Notes8_1.htm#:~:text=T%20%3D%20T0%20%2Fta