

IF3170 Inteligensi Artifisial

Tugas Besar 1: Implementasi Algoritma Pembelajaran Mesin



Disusun Oleh:

13522016 Zachary Samuel Tobing

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2024**

BAB I

DESKRIPSI PERSOALAN

Pembelajaran mesin merupakan salah satu cabang dari kecerdasan buatan yang memungkinkan sistem untuk belajar dari data dan membuat prediksi atau keputusan tanpa diprogram secara eksplisit.

Dataset **UNSW-NB15** adalah kumpulan data lalu lintas jaringan yang mencakup berbagai jenis serangan siber dan aktivitas normal. Pada tugas ini, Anda diminta untuk mengimplementasikan algoritma pembelajaran mesin yang telah kalian pelajari di kuliah, yaitu **KNN**, **Gaussian Naive-Bayes**, dan **ID3** pada dataset **UNSW-NB15**.

Untuk menghasilkan prediksi yang berkualitas, Anda diharuskan untuk melakukan beberapa tahap berikut ini (tahapan lebih lengkap dapat dilihat di template notebook):

Data Cleaning

Tahap ini bertujuan untuk membersihkan dataset dari nilai yang hilang (missing values), data duplikat, atau data yang tidak valid sehingga dataset siap digunakan untuk analisis.

Data Transformation

Transformasi data melibatkan langkah-langkah seperti encoding variabel kategori, normalisasi atau standarisasi fitur numerik, serta penanganan ketidakseimbangan data (imbalanced data) untuk memastikan data berada dalam format yang sesuai dengan algoritma pembelajaran mesin.

Feature Selection

Pemilihan fitur yang relevan bertujuan untuk mengurangi kompleksitas model, menghindari overfitting, serta meningkatkan kinerja model. Langkah ini melibatkan identifikasi fitur yang memiliki pengaruh signifikan terhadap variabel target.

Dimensionality Reduction

Jika dataset memiliki jumlah fitur yang besar, reduksi dimensi dapat digunakan untuk mengurangi dimensi tanpa kehilangan informasi penting. Teknik seperti Principal Component Analysis (PCA) sering digunakan pada tahap ini.

Modeling dan Validation

Pada tahap ini, algoritma pembelajaran mesin seperti K-Nearest Neighbors (KNN), Naive Bayes, dan ID3 (Iterative Dichotomiser 3) diterapkan pada dataset. Anda akan melatih model pembelajaran mesin yang akan **mengklasifikasi kategori attack (attack_cat)** berdasarkan fitur-fitur lain yang telah diberikan. Model yang telah dibuat divalidasi menggunakan metode seperti **train-test split** atau **k-fold cross-validation** untuk memastikan kinerja yang optimal.

BAB II

IMPLEMENTASI

2.1 Pemilihan Objective Function

Sebagai kriteria dari *diagonal magic cube*, jumlah semua angka pada setiap baris, kolom, dan tiang harus sama dengan *magic constant* (kriteria *magic cube*), ditambah dengan jumlah semua diagonal pada setiap lapisan harus berjumlah sama juga dengan *magic constant*.

Oleh karena itu, *objective function* yang dirancang perlu untuk memeriksa kriteria ini, sehingga dipilih fungsi objektif yaitu **jumlah setiap garis** (baris, kolom, tiang, diagonal bidang, diagonal ruang) yang memenuhi kriteria tersebut, yaitu **berjumlah sama dengan *magic constant***.

Magic constant merupakan suatu konstanta yang sama untuk setiap orde *magic cube*, diperoleh dari rumus:

$$M_3(n) = (n(n^3 + 1)) / 2$$

Sehingga dapat dihitung untuk orde 5 pada persoalan yaitu:

$$M_3(5) = (5(5^3 + 1)) / 2 = (5 \times 126) / 2 = 315$$

Total jumlah garis yang ditinjau pada suatu *diagonal magic cube* berorde m adalah $3m^2 + 6m + 4$, diperoleh dari:

- m^2 untuk setiap garis lurus (baris, kolom, tiang)
- 2 untuk setiap sisi, untuk 3 arah (baris, kolom, tiang), m untuk banyaknya setiap arah
- 4 untuk diagonal ruang (konstan)

Melihat rumus tersebut, maka dapat dihitung untuk *diagonal magic cube* orde 5 adalah:

$$3(5)^2 + 6(5) + 4 = 75 + 30 + 4 = 109$$

Oleh karena itu, nilai objektif maksimum suatu *state* adalah ketika *magic cube* sudah berada dalam solusi (semua kriteria terpenuhi) adalah 109, dan nilai objektif minimum suatu *state* adalah 0, ketika tidak ada garis yang berjumlah sama dengan *magic constant*.

2.2 Algoritma Local Search

Model *local search* merupakan sebuah metode optimasi pencarian dalam lingkup kompleks untuk menyelesaikan permasalahan komputasi. Metode *local search* merupakan metode yang tidak bergantung pada rangkaian, sehingga urutan pada kasus ini tidak penting, tetapi kondisi awalnya dinyatakan dalam konfigurasi lengkap. Metode ini mencari kondisi optimal dengan bergerak dari kondisi saat ini (*current state*) ke kondisi selanjutnya, di mana *neighbors* atau tetangganya memiliki nilai yang lebih baik dibandingkan saat ini. Tujuan dari metode ini adalah untuk mencari nilai yang merupakan puncak tertinggi (*global optimum*), akan tetapi tidak jarang metode ini terhenti pada nilai *local optimum*. Terdapat beberapa algoritma *local search* yang biasa digunakan, sebagai berikut.

- Steepest Ascent Hill-climbing

Steepest Ascent Hill-climbing merupakan algoritma *local search* di mana hanya terdapat satu buah *current state* untuk setiap iterasi. Metode ini membangkitkan semua suksesor yang mungkin. Kondisi *current state* hanya akan berubah jika nilai dari tetangganya (*neighbors*) lebih baik atau setidaknya sama dengan nilai dari *current state*. Algoritma ini akan berhenti ketika *current state* sudah mencapai *peak* atau *flat* di mana tidak lagi terdapat *neighbor* yang memiliki nilai lebih tinggi jika dibandingkan dengan *current state*. Metode ini bekerja sangat cepat, tetapi tingkat keberhasilannya untuk mencapai *global optimum* sangat rendah dan biasanya terhenti pada *local optimum*.

- Hill-climbing with Sideways Move

Hill-climbing with Sideways Move merupakan variasi dari algoritma “Steepest Ascent Hill-climbing”. Seperti algoritma “Steepest Ascent Hill-climbing”, metode ini membangkitkan semua suksesor yang mungkin, yang nilainya hanya akan berubah jika nilai dari tetangganya lebih baik dari kondisi saat ini. Namun, algoritma “Sideways Move” tidak memperbolehkan perpindahan jika nilai dari *neighbors* sama atau bahkan lebih kecil dibandingkan kondisi saat ini. Algoritma ini hanya akan berhenti jika *current state* telah mencapai *peak* dengan pengulangan pada limitasi tertentu. Metode ini bekerja

sedikit lebih lambat dibandingkan dengan “Steepest Ascent Hill-climbing”, tetapi dengan tingkat keberhasilan yang jauh lebih besar.

- Random Restart Hill-climbing

Random Restart Hill-climbing merupakan variasi dari algoritma “Steepest Ascent Hill-climbing”. Metode ini semua suksesor yang mungkin pada *initial state*-nya. Sama seperti “Steepest Ascent Hill-climbing”, metode ini juga hanya akan mengubah kondisi *current state*-nya jika nilai dari tetangganya lebih baik atau setidaknya sama dengan kondisi saat ini. Perbedaan metode ini dengan “Steepest Ascent Hill-climbing” adalah, metode ini akan selalu mencoba mencari nilai *global optimum* hingga mencapai limitasi berupa waktu dimana algoritma ini berjalan dan jumlah limitasi pengulangan yang dapat dilakukan.

- Stochastic Hill-climbing

Stochastic Hill-climbing merupakan variasi dari algoritma “Steepest Ascent Hill-climbing” di mana, *agent* hanya membangkitkan sebuah nilai *current state* secara acak. Seperti metode hill-climbing lainnya, metode ini juga hanya akan mengubah kondisi saat ini jika nilai tetangganya lebih baik dibandingkan *current state*. Stochastic Hill-climbing akan berhenti bekerja saat sudah mencapai limitasi jumlah iterasi yang dapat dilakukan. Metode “Stochastic Hill-climbing” termasuk kedalam metode yang lambat, karena diharuskan melakukan banyak *step* tambahan.

- Simulated Annealing

Simulated Annealing (SA) merupakan kombinasi antara metode “hill-climbing” yang efisien dengan metode “random walk” yang lebih menyeluruh. Ide dari metode ini adalah memperbolehkan adanya perpindahan dari kondisi saat ini ke kondisi yang lebih buruk, dengan catatan nilai dari kondisi yang lebih buruk ini akan semakin menurun selama iterasi berlangsung. Sama seperti “Stochastic Hill-climbing” metode ini hanya membangkitkan satu suksesor secara acak. Metode ini memungkinkan dua buah skenario untuk berpindah dari satu kondisi ke kondisi lainnya, yaitu nilai tetangganya lebih baik dibanding nilai keadaan sekarang atau dapat pula dengan memperhitungkan probabilitas perpindahan jika nilai tetangganya lebih buruk dibandingkan kondisi saat ini. Probabilitas perpindahannya mengikut persamaan $e^{\Delta E/T}$, dengan ΔE merupakan

perbedaan antara nilai tetangga dengan nilai saat ini. Metode ini akan berhenti jika nilai T telah mendekati nol.

- Genetic Algorithm

Genetic Algorithm (GA) merupakan sebuah algoritma *local search* setiap individu dianggap sebagai sebuah string dengan *finite alphabet*. Tidak seperti metode-metode *local search* yang telah disebutkan di atas, metode “Genetic Algorithm” perlu melakukan beberapa persiapan terlebih dahulu sebelum menjalankan algoritma. Pada awalnya akan dihasilkan *initial population* yang menjadi *parent* pada metode ini dengan membangkitkan secara acak k states . Kemudian, dilakukan *fitness function*. Setelahnya, akan dipilih suksesor berdasarkan *fitness function* yang ada. Suksesor pada algoritma GA dipilih dengan mengkombinasikan dua buah *parent*. Suksesor yang telah dipilih akan dilakukan *crossover* dengan *output* dari *parent* lainnya. Barulah pada akhirnya, akan dilakukan mutasi untuk menghasilkan individu-individu baru.

2.3 Implementasi Algoritma Local Search

Berdasarkan spesifikasi tugas besar, algoritma *local search* yang digunakan berupa *hill-climbing with sideways move*, *simulated annealing*, dan *genetic algorithm*. Program ini menggunakan bahasa *Python* untuk logika dasarnya dan menggunakan pendekatan OOP (*Object Oriented Programming*). Berikut penjelasan mengenai kelas dan metode yang digunakan untuk mengimplementasikan programnya.

2.3.1 Cube

Untuk mengimplementasikan program, kami membuat sebuah kelas Cube yang mana kelas ini akan merepresentasikan sebuah *magic cube* 3D dengan dimensi $n \times n \times n$. Kelas ini memiliki atribut `size` dengan nilai default $n = 5$. Pada kelas ini juga terdapat atribut `total_elements`, yang menyimpan jumlah total elemen dalam cube, dengan total n^3 . Sementara itu, `magic_constants` merupakan nilai yang dihitung berdasarkan rumus khusus untuk *magic cube*. Cube itu sendiri disimpan dalam atribut `__cube`. Fungsi `initialize_cube` menginisialisasi elemen-elemen dari *magic cube* yang menyusun tiap layer secara acak, *magic cube* yang diinisiasikan pada awal ditampilkan dalam bentuk elemen-elemen untuk setiap layer. Sementara itu, untuk menentukan nilai *objective function* digunakan fungsi `evaluate_fitness`.

Dengan diperbolehkan untuk melakukan beberapa langkah berupa menukar posisi dua buah angka, dibuat sebuah fungsi `random_neighbor` yang dapat mengubah posisi dua buah angka secara acak.

2.3.2 HillClimbingSidewaysMove

Implementasi metode algoritma Hill-climbing with Sideways Move memerlukan function `initializeCube` untuk men-generate *diagonal magic cube* pertama dan disimpan dalam atribut `self.cube`. Setiap iterasi beserta nilai *objective function*, masing-masing disimpan dalam atribut `self.iterations` dan `self.fitness_values`. Batasan *sideways move* yang diizinkan secara default adalah 100. Nilai *sideways move* akan ditambahkan setiap iterasi di mana *nilai objective function saat ini* \leq *nilai objective function neighbor*. Saat telah mencapai batasan *sideways move* program akan berhenti mencari solusi permasalahan. Pada akhir program, function `plotResults` akan menampilkan plot iterasi terhadap nilai *objective function*.

2.3.3 SimulatedAnnealing

Implementasi algoritma *local search* Simulated Annealing (SA) function `initializeCube` untuk men-generate *diagonal magic cube* pertama dan disimpan dalam atribut `self.cube`. Atribut `self.delta` menyimpan besar perbedaan antara nilai *objective function* saat ini dengan nilai *objective function neighbor*, sedangkan atribut `self.temperatures` merupakan nilai bertindak sebagai *controller* dalam eksplorasi pada algoritma Simulated Annealing, nilai temperatur ini akan secara bertahap menurun dengan memanfaatkan function `schedule`. Saat nilai temperatur mendekati akhir, algoritma Simulated Annealing akan memiliki perilaku yang mirip dengan algoritma Hill Climbing. Algoritma ini akan berhenti ketika temperatur bernilai nol, atau pada saat nilai pada atribut `self.local_optima_stuck_count` telah mencapai nilai atribut `self.stuck_threshold`.

2.3.4 GeneticAlgorithm

Pada implementasi algoritma *local search* Genetic Algorithm (GA) dilakukan pembangkitan elemen *magic cube* secara acak dengan menggunakan function

`initialize_population`. Dengan menggunakan function `crossover`, *magic cube* yang telah dibangkitkan akan dilakukan *crossover* atau persilangan dengan populasi lainnya. Produk hasil *crossover* dilakukan *mutation* dengan function `mutation`, yang mana akan ditentukan hasilnya melalui function `selection_wheel`, hasil ini akan diurutkan berdasarkan *fitness value*-nya pada function `sort_fitness_value`.

Heuristik yang digunakan pada *crossover* yaitu penukaran nilai-nilai pada suatu *layer* saja dikarenakan adanya kemungkinan bahwa nilai yang ditukar sama sehingga jika dilakukan pada banyak bagian akan menghasilkan *magic cube* dengan banyak nilai duplikat.

Heuristik yang digunakan pada *mutation* yang dilakukan juga berupa *swapping* antara nilai pada suatu kubus pada dua buah titik *random* sebanyak 3 kali agar menghasilkan variasi kubus yang besar dan mencegah nilai-nilai duplikat pada kubus.

Selection wheel yang digunakan berupa *lucky spin*.

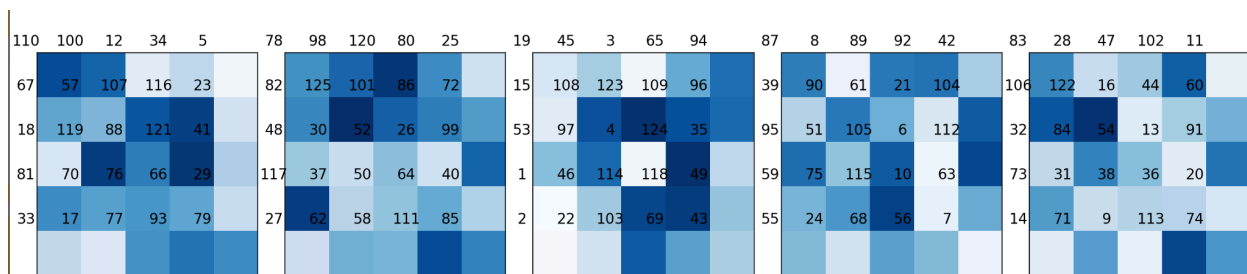
2.3.5 Visualizer

Pada *class* ini *magic cube* yang ada divisualisasikan dengan perangkat *numpy* dan *matplotlib* dengan warna berupa representasi nilainya.

2.4 Eksperimen

Berikut berupa hasil dari eksperimen sesuai dengan ketentuan spesifikasi tugas berdasarkan algoritmanya.

2.4.1 Hill-Climbing with Sideways move



3 maximum sideways

Layer 1:

[117, 114, 52, 15, 38]

[26, 69, 90, 57, 4]

[43, 28, 96, 9, 29]

[27, 121, 91, 31, 82]

[105, 73, 11, 34, 44]

Layer 2:

[61, 49, 36, 50, 81]

[67, 75, 18, 112, 120]

[122, 17, 33, 21, 125]

[71, 107, 25, 41, 76]

[19, 35, 68, 51, 109]

Layer 3:

[32, 58, 103, 83, 7]

[89, 30, 106, 20, 119]

[47, 42, 124, 1, 70]

[102, 92, 80, 98, 16]

[60, 46, 79, 123, 95]

Layer 4:

[87, 97, 115, 13, 66]

[84, 72, 59, 116, 77]

[6, 37, 40, 94, 86]

[108, 104, 85, 2, 88]

[22, 93, 53, 110, 100]

Layer 5:

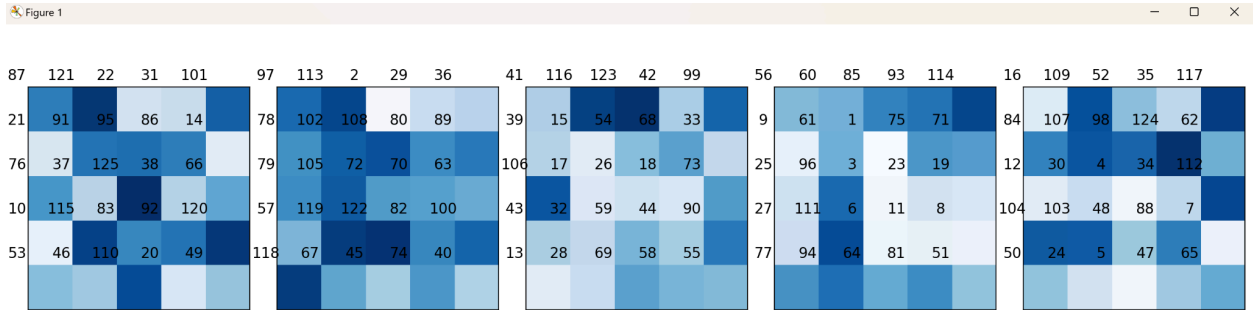
[62, 56, 23, 99, 3]

[12, 113, 111, 24, 65]

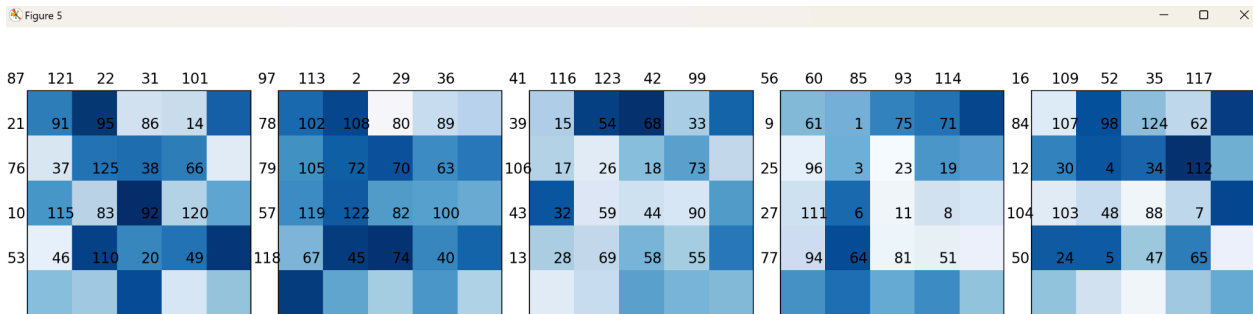
[8, 101, 74, 48, 39]

[5, 63, 118, 10, 54]

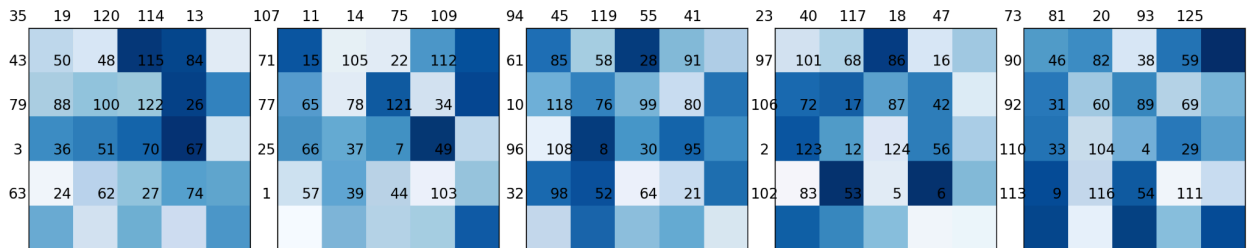
[45, 64, 78, 14, 55]



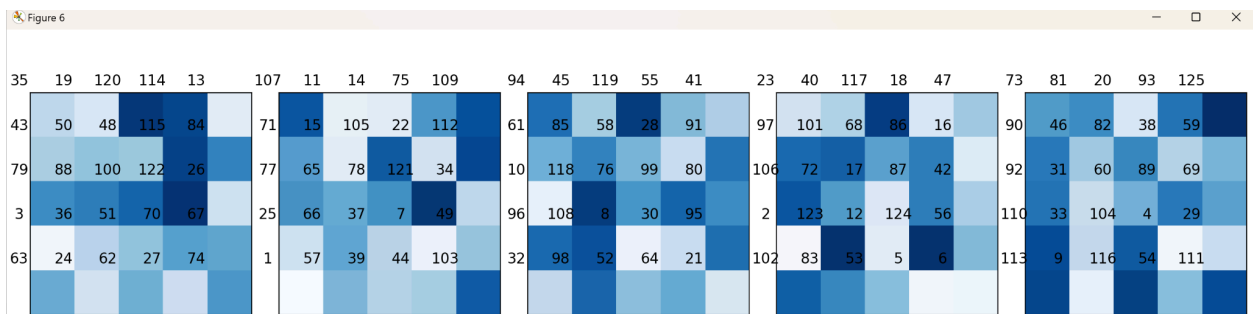
Maximum sideways move 2



Final



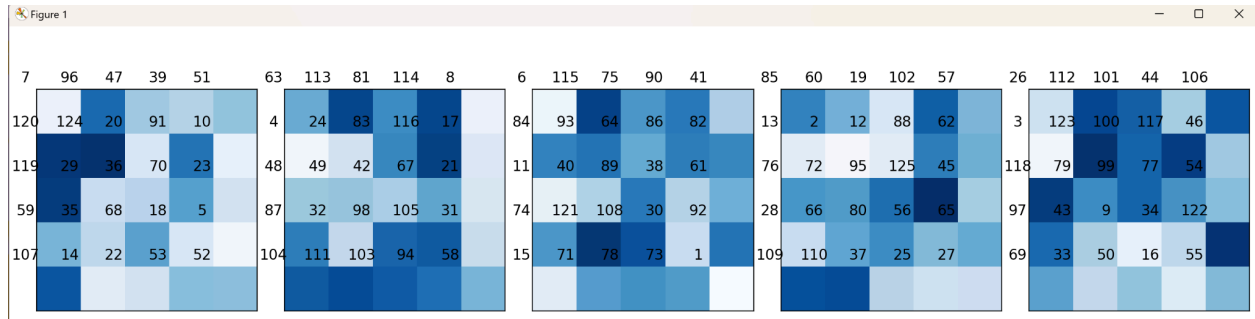
Maximum sideways move 1



Final

Duration: 0.6485280990600586
Number of Iterations: 2

2.4.2 Simulated Annealing (iterasi yang sangat banyak)




 Figure 5

 Figure 6

 Figure 7

 Figure 8

 Figure 9


 Figure 10

 Figure 11

 Figure 12






 Figure 13


 Figure 14


 Figure 15


 Figure 16


 Figure 17


 Figure 18


 Figure 19


 Figure 20

 Figure 21

 Figure 22

 Figure 23

 Figure 24

 Figure 25

2.4.3 Genetic Algorithm (sempat dicoba pada terminal)

2.4.4 Stochastic (Tidak sempat dicoba)

2.5 Analisis

Berdasarkan hasil eksperimen yang diperoleh pada bagian 2.4, dilakukan analisis pada hasil setiap algoritma sebagai berikut.

2.5.1 Hill-Climbing with Sideways move

Dapat dilihat bahwa algoritma *sideways* tidak menghasilkan solusi yang baik dikarenakan sifatnya yang *stuck* pada lokal optimum sehingga nilainya tidak berubah banyak dari nilai awalnya.

2.5.2 Simulated Annealing

Dapat dilihat pada *simulated annealing* bahwa jawaban yang diberikan juga cukup jauh dibandingkan dengan nilai objektif maksimum (solusi). Ini dikarenakan sifat algoritmanya yang banyak mengandung nilai acak sehingga tidak mengarah ke solusi, meskipun memiliki jumlah iterasi yang besar.

2.5.3 Genetic Algorithm

Dapat diamati bahwa solusi yang diberikan oleh *genetic algorithm* lebih baik dengan nilai objektif rata-rata yang lebih tinggi dibandingkan algoritma lain. Ini dikarenakan meskipun seringkali algoritma ini memiliki banyak nilai duplikat karena heuristik yang digunakan, ini memberikan variasi yang lebih besar sehingga lebih banyak kemungkinan penjumlahan serta kombinasi yang jumlahnya serupa dengan *magic constant*.

BAB III

CLEANING DAN PREPROCESSING

3.1 Kesimpulan

Dapat disimpulkan bahwa sebaik apapun heuristik yang digunakan, algoritma *local search* tidak mendekati solusi. Jika ingin dibandingkan antar algoritma, maka algoritma yang terbaik adalah *genetic algorithm* karena menghasilkan lebih banyak variasi populasi pada setiap iterasinya dengan batasan iterasi yang dapat ditentukan oleh pengguna.

Dapat dilihat bahwa algoritma *hill climbing with sideways move* tidak berperforma dengan baik karena *stuck* pada lokal *optimum*. Ketika heuristik yang digunakan adalah *swapping* dibandingkan hanya merubah suatu nilai secara acak, dapat dilihat bahwa hasilnya tidak terlalu baik.

Untuk *simulated annealing* dapat juga dilihat karena cara kerjanya yang menggunakan *random neighbor*, hasil yang diberikan juga tidak baik.

3.2 Saran

Sebaiknya program yang dibuat dikembangkan lebih baik lagi agar memiliki heuristik dan tampilan yang lebih baik.

BAB IV

ANALISIS

BAB V

KESIMPULAN DAN SARAN

BAB V

REFERENSI

Russell, S., & Norvig, P. (2016). *Artificial intelligence: A Modern Approach, Global Edition*.