Analisis Penerapan Algoritma Greedy pada Sistem Lampu Lalu Lintas Otomatis

Dengan Mencari Algoritma Greedy Terbaik untuk Mengurai Kemacetan

Keanu Amadius Gonza Wrahatno - 13522082
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13522082@std.stei.itb.ac.id

***Abstrak*— Kemacetan lalu lintas merupakan masalah yang menyebabkan pemborosan waktu dan bahan bakar. Sistem lampu lalu lintas otomatis dapat menyelesaikan masalah ini dengan algoritma *greedy*. Algoritma *greedy* akan membuat keputusan optimal lokal untuk menentukan jalur yang diberi lampu hijau berdasarkan panjang antrean kendaraan secara *real-time*. Penggunaan Algoritma Greedy juga memungkinkan kita untuk memperbarui data antrean jalur di tiap interval waktu tertentu. Makalah ini mengkaji penerapan algoritma *greedy* untuk mengurangi kemacetan. Analisis mencakup konsep dasar algoritma *greedy*, implementasi dalam sistem lampu lalu lintas, serta manfaat dan tantangan dalam penerapannya.**

***Kata kunci*—*Greedy*, Lampu Lalu Lintas Otomatis, Kemacetan, Strategi Algoritma**

I. PENDAHULUAN

Kemacetan lalu lintas menjadi masalah yang umum terjadi di kota-kota besar. Kemacetan dapat menyebabkan berbagai masalah lainnya seperti pemborosan waktu dan bahan bakar. Masalah lain yang dapat ditimbulkan adalah polusi udara, polusi suara, dan tingkat stres pengemudi. Hal seperti ini dapat diatasi dengan menggunakan sistem lampu lalu lintas otomatis.

Lampu lalu lintas otomatis merupakan sebuah sistem yang dapat mengatur persimpangan jalan secara efisien berdasarkan data asli banyaknya kendaraan di suatu jalur saat itu. Sistem ini akan mengendalikan persimpangan dengan menentukan jalur mana yang harus diberi jalan serta menentukan berapa lama durasi lampu merah, kuning, dan hijaunya. Dengan begitu, sistem ini dapat digunakan untuk mengurangi kemacetan dan mengurangi waktu tunggu kendaraan di suatu persimpangan dengan. Hal ini dikarenakan sistem lampu lalu lintas otomatis akan mendahulukan jalur yang memiliki antrean paling panjang dan data antrean dapat diperbaharui selalu karena tiap jalur pasti akan mengalami penambahan antrean.

Pada makalah ini, penulis akan mengaplikasikan ilmu Strategi Algoritma, yaitu algoritma *Greedy* dalam sistem lampu lalu lintas otomatis untuk mengurangi kemacetan. Algoritma ini akan mengambil optimum lokal dengan harapan memberi solusi yang optimal secara keseluruhan. Penerapan algoritma *greedy* pada sistem lampu lalu lintas otomatis diharapkan dapat memberikan solusi yang lebih cepat dan efisien untuk mengurangi kemacetan dibandingkan dengan metode konvensional yang tidak memiliki algoritma atau waktunya statis.

II. LANDASAN TEORI

A. Algoritma Greedy

Algoritma *Greedy* merupakan algoritma yang dirancang untuk mengambil keputusan menurut kondisi saat itu juga atau mencari optimum lokal pada setiap langkah dengan harapan untuk dapat menemukan solusi global yang optimum. Algoritma *Greedy* akan berusaha untuk menemukan solusi dalam waktu yang sesingkat mungkin.

Algoritma *Greedy* termasuk dalam jenis algoritma optimasi. Hanya ada dua macam persoalan optimasi yaitu:

- Maksimasi
- Minimasi

Ada banyak pilihan yang perlu dievaluasi pada setiap langkahnya sebelum melanjutkan ke langkah berikutnya. Pada setiap langkah harus diambil keputusan yang terbaik karena algoritma ini tidak dapat kembali ke langkah sebelumnya untuk mengambil langkah lain. Sehingga kita dapat menggunakan penentuan optimum lokal untuk menyelesaikan ini.

Pemecahan persoalan dilakukan dengan ketentuan pada tiap langkahnya:

- Mengambil pilihan terbaik pada saat itu juga tanpa memperhatikan konsekuensi ke depannya. Hal ini menerapkan prinsip “ambil apa yang bisa kamu ambil saat ini juga”
- Berharap bahwa dengan memilih optimum lokal pada setiap langkahnya dapat membawa ke optimum global diakhir. Hal ini berarti untuk menemukan solusi yang optimal pada akhir algoritmanya

Elemen-elemen yang ada pada Algoritma *Greedy* yaitu:

1. Himpunan Kandidat (C)
   Merupakan himpunan yang berisi kandidat yang akan dipilih pada setiap langkahnya
2. Himpunan Solusi (S)
   Merupakan himpunan yang berisi kandidat yang sudah dipilih.
3. Fungsi Solusi
   Fungsi yang menentukan apakah himpunan kandidat yang dipilih sudah memberikan solusi.
4. Fungsi Seleksi
   Fungsi untuk memilih kandidat berdasarkan strategi *greedy* tertentu.
5. Fungsi Kelayakan
   Fungsi yang memeriksa apakah kandidat yang dipilih

layak atau tidak layak untuk dimasukkan ke dalam himpunan solusi

6. Fungsi Obyektif
   Fungsi yang memaksimumkan atau fungsi yang meminimumkan

Algoritma *greedy* akan melakukan pencarian sebuah himpunan bagian S, dari himpunan kandidat C. S harus memenuhi kriteria yang ditentukan atau S menyatakan suatu solusi dan S dioptimisasi oleh fungsi obyektif.

```
function greedy(C : himpunan_kandidat) → himpunan_solusi
{ Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy }
Deklarasi
   x : kandidat
   S : himpunan_solusi

Algoritma:
   S ← {}      { inisialisasi S dengan kosong }
   while (not SOLUSI(S)) and (C ≠ {} ) do
      x ← SELEKSI(C)     { pilih sebuah kandidat dari C}
      C ← C – {x}        { buang x dari C karena sudah dipilih }
      if LAYAK(S ∪ {x}) then      { x memenuhi kelayakan untuk dimasukkan ke dalam himpunan solusi }
         S ← S ∪ {x}      { masukkan x ke dalam himpunan solusi }
      endif
   endwhile
   {SOLUSI(S) or C = {} }

   if SOLUSI(S) then    { solusi sudah lengkap }
      return S
   else
     write('tidak ada solusi')
   endif
```

Gambar 1. Skema Umum Algoritma Greedy
(Sumber : [1])

Perlu diperhatikan bahwa optimum global yang dicapai belum tentu merupakan solusi yang terbaik karena bisa jadi merupakan solusi *sub-optimum* atau *pseudo-optimum*. Hal ini karena algoritma *greedy* tidak melakukan pencarian ke semua kemungkinan solusi yang ada dan ada fungsi seleksi yang berbeda sehingga kita harus mendefinisikan fungsi seleksi yang tepat agar algoritma ini dapat menghasilkan solusi yang optimal.

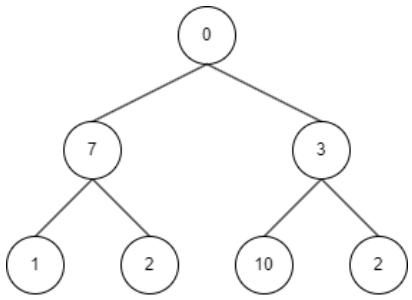
Algoritma *greedy* dapat digunakan untuk menghasilkan solusi hampiran untuk sebuah persoalan yang solusi terbaik mutlaknya tidak perlu diperhatikan. Namun, bila persoalan bisa didapat solusi optimalnya, maka harus bisa dibuktikan secara matematis.

Berikut ini merupakan ciri-ciri yang bisa dilihat dari algoritma *greedy*:

1. Algoritmanya menyelesaikan masalah dengan mencari solusi optimal yang bisa *minimasi* ataupun *maksimasi*. Hal ini dilakukan dengan mencari optimum lokal atau menentukan pilihan berdasarkan pilihan yang tersedia sekarang.
2. Algoritmanya cepat dan efisien dengan waktu kompleksitasnya O(n log n) atau O(n) sehingga dapat diterapkan kedalam masalah yang besar.
3. Algoritma ini berjalan satu kali. Dalam kata lain pencarian solusi optimal dilakukan tanpa pengulangan karena algoritma ini tidak dapat mundur. Sehingga semua kemungkinan tidak di periksa dan tidak memeriksa kebenaran hasil yang dihasilkan
4. Mudah untuk diterapkan

B. Greedy VS Non-Greedy

Misalkan ada sebuah grafik sebagai berikut:



Gambar 2. Skema Umum Algoritma Greedy
(Sumber : Arsip Pribadi)

Akan dicari jalur dengan penjumlahan bobot angka di jalur tersebut paling besar. Jika menggunakan algoritma *non-greedy*, maka yang dihasilkan yaitu 0,3,10 dengan total bobotnya yaitu 13. Jika menggunakan *greedy* maka pada langkah pertama akan mengambil 7 dimana sampai saat ini solusi yang dihasilkan optimal. Namun jika lanjut ke langkah berikutnya, algoritma akan memilih 2 yang mana penjumlahan jalurnya (0,7,2) hanya 9.

Ini merupakan kelemahan dari algoritma *greedy* sehingga pastikan bahwa algoritma *greedy* hanya dipakai jika suatu masalah memerlukan pilihan terbaik saat ini dan masalah tersebut memerlukan solusi optimal baik minimum atau maksimum

C. Lampu Lalu Lintas

Lampu lalu lintas digunakan untuk mengatur sebuah persimpangan jalan agar pengendara tertib dan menghindari adanya kecelakaan. Warna merah menandakan berhenti, warna kuning menandakan hati-hati, dan warna hijau menandakan jalan.

Lampu lalu lintas di Indonesia masih menggunakan metode konvensional dimana pemberian warna lampu tidak memerlukan algoritma melainkan menggunakan waktu yang statis. Hal ini memiliki banyak kekurangan yaitu tidak dapat mengurai kemacetan secara efisien. Waktu yang statis berarti sistem ini tidak memedulikan apa pun sebagai aspek dalam pemberian waktunya. Oleh karena itu, apabila suatu jalur sudah kosong, akan ada kemungkinan jalur tersebut masih diberi warna hijau atau jika ada jalur yang memiliki antrean panjang hanya diberi waktu statis yang tidak sesuai antrean.

Lampu lalu lintas dengan sistem otomatis diharapkan dapat mengurai masalah kemacetan yang ditimbulkan oleh sistem konvensional. Dengan ditambahkannya algoritma, pemberian warna hijau dapat diberikan ke jalur yang sesuai. Misalnya adalah jalur yang memiliki antrean sangat panjang, maka akan diberi waktu hijau yang lama atau jika jalur tersebut kosong maka tidak perlu diberi lampu hijau.

Dengan disematkannya algoritma , membuat waktu lampu berwarna hijau tidak statis lagi. Pemberian waktu warna hijau didasari dari data *real-time* berdasarkan prioritas kondisi antrean saat ini di suatu jalur. Semakin panjang antreannya, maka akan semakin lama pula lampu hijaunya.

Pastinya perlu juga adanya batasan-batasan yang bisa menyempurnakan kondisi ini. Misalnya jika ada jalur yang hanya ada 1 mobil. Maka jalur tersebut tidak akan diberikan

lampu hijau jika jalur-jalur lainnya ada banyak mobil. Untuk mengatasi masalah ini, harus ada batasan jika sudah sampai waktu sekian, maka jalur tersebut diberi lampu hijau untuk sekian menit. Sistem ini dapat membuat waktu tunggu rata-rata kendaraan berkurang dan mengurai kemacetan

III. PEMBAHASAN

A. Pemilihan Greedy

Pada masalah penguraian kemacetan, strategi yang dipilih untuk menyelesaikan ini adalah algoritma *greedy*. Sistem lampu lalu lintas otomatis tidak memperhatikan solusi terbaik yang mutlak sehingga algoritma *greedy* dapat dipakai untuk mendapatkan solusi hampiran /aproksimasi.

Greedy dipilih karena sistem lampu otomatis memerlukan keputusan saat itu juga dan tidak memikirkan konsekuensi kedepannya. Misal saat ini ada 10 mobil di jalur A dan 2 mobil di jalur B, maka algoritma akan memberi lampu hijau ke jalur A.

Greedy juga dipilih pada penyelesaian masalah sistem lampu lalu lintas karena data pada tiap jalur dapat berubah atau diperbarui tiap interval waktu tertentu. Jika data diperbarui, maka algoritma greedy akan tetap lanjut dengan pilihan keputusannya tanpa perlu mundur lagi. Ini yang dibutuhkan dalam sistem lampu lalu lintas otomatis.

B. Sistem lampu Lalu Lintas Otomatis

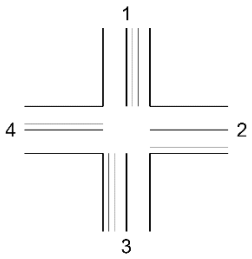
Secara garis besar, proses penerapan sistem lampu lalu lintas otomatis dengan batasan batasan tertentu sebagai berikut :

- Algoritma akan memilih jalur dengan bobot terbesar. Hal ini merupakan persoalan maksimasi dalam algoritma greedy.
- Setiap jalur ada 4 baris. Baris paling kiri untuk mobil yang belok kiri, 2 baris tengah untuk mobil lurus, dan 1 baris paling kanan untuk mobil belok kanan.
- Setiap jalur yang dipilih akan merepresentasikan pemberian lampu hijau selama 60 detik.
- Anggap laju mobil yang keluar dari antrean jalur yaitu 0,333 mobil / detik untuk jalur lurus dan 0,16 mobil / detik untuk jalur belok kanan. Maka tiap jalur yang diberi lampu hijau akan mengurangkan 20 mobil untuk jalur lurus dan 10 mobil untuk jalur belok kanan.
- Bobot suatu jalur dihitung dengan rumus mobil lurus + 2 * mobil belok kanan. Mobil belok kanan dikali 2 karena hanya memiliki 1 jalur dan akan menyebabkan antrean lebih panjang daripada mobil yang akan lurus.
- Jalur yang telah dipilih akan dikurangkan bobotnya sesuai ketentuan sebelumnya lalu dimasukkan lagi ke dalam himpunan kandidat dengan bobot yang baru. Sehingga hal ini memungkinkan jalur tersebut dipilih 2 kali.
- Jika jalur tersebut dipilih 2 kali, maka direpresentasikan bahwa jalur tersebut diberikan waktu sebesar 2 x 80 = 160 detik.
- Diberikan batasan apabila ada suatu jalur yang tidak dipilih dalam 480 detik, maka jalur tersebut akan didahulukan / diberikan lampu hijau selama 20 detik.

C. Identifikasi Elemen Greedy

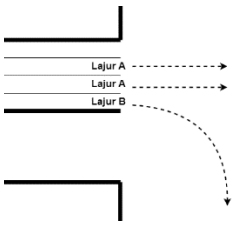
Sistem lampu lalu lintas otomatis dapat diidentifikasikan menurut elemen-elemen pada algoritma *greedy* sebagai berikut:

1. Himpunan Kandidat (C)
   Himpunan kandidat pada masalah ini yaitu jalur jalur pada persimpangan. Misalkan ada 4 jalur yaitu jalur 1,2,3, dan 4 seperti di bawah ini



Gambar 3. Persimpangan 4 jalur
(Sumber : Arsip Pribadi)

   Sehingga himpunan kandidatnya yaitu jalur {1,2,3,4}
   Tiap jalur menyimpan banyaknya mobil yang akan lurus dan banyaknya mobil yang akan belok kanan.
2. Himpunan Solusi (S)
   Himpunan solusi pada masalah ini yaitu urutan jalur yang dihasilkan untuk mengurai kemacetan.
3. Fungsi Solusi
   Dikarenakan persoalan ini tidak punya solusi terbaik yang mutlak, fungsi solusi pada persoalan ini dicek pada tiap langkahnya apakah telah memilih langkah yang paling besar.
4. Fungsi Seleksi
   Fungsi seleksi pada persoalan ini ditentukan dengan menghitung bobot jalur terbesar. Tiap jalurnya mempunyai 3 lajur, anggap lajur paling kiri untuk belok kiri dan tidak perlu mengikuti lampu sehingga kita hanya melihat 3 lajur ini saja.



Gambar 4. Lajur di suatu jalur
(Sumber : Arsip Pribadi)

   Sehingga rumus untuk menghitung bobot suatu jalur yaitu:

$$Bobot = (jumlah\ mobil\ lurus) + (jumlah\ mobil\ belok\ kanan) * 2$$

   Jumlah mobil belok kanan dikali 2 karena antreannya hanya 1 lajur sehingga memungkinkan kemacetan lebih besar. Maka dari itu bobot ini dikali 2 agar panjang antrean seimbang.
   Setelah jalur dipilih, kurangi kapasitas mobil pada lajur dan masukan kembali ke himpunan kandidat. Dengan ketentuan pengurangan 20 mobil yang akan lurus dan pengurangan 10 mobil yang belok kanan.

5. Fungsi Kelayakan
   Banyak sekali fungsi kelayakan yang harus didefinisikan pada persoalan ini untuk meningkatkan efisiensi.
   Jika suatu jalur tidak diberi lampu hijau selama 6 iterasi, maka untuk langkah selanjutnya yang akan dipilih adalah jalur tersebut.
   Untuk menstimulasikan data real dimana mobil akan terus berdatangan, maka dibuat proses masukan *input user* tiap 4 iterasi untuk *update* tambahan mobil yang masuk ke jalur.
6. Fungsi Obyektif
   Fungsi Obyektif dalam persoalan ini adalah memaksimalkan bobot yang ada pada jalur dengan mempertimbangkan kapasitas mobil yang akan lurus dengan mobil yang akan belok kanan

IV. IMPLEMENTASI

Implementasi dilakukan dengan membuat kelas jalur terlebih dahulu. Kelas jalur akan memuat informasi berupa id, banyaknya mobil yang belok_kanan, banyaknya mobil yang lurus, dan kapan ia dipilih terakhir untuk menentukan jika sudah 6 iterasi tidak dipilih, maka ia akan terpilih.

```
public class Jalur {
    int id;
    int belok_kanan;
    int lurus;
    int lastSelected;

    public Jalur(int id, int belok_kanan, int
lurus) {
        this.id = id;
        this.belok_kanan = belok_kanan;
        this.lurus = lurus;
        this.lastSelected = 0;
    }

    public int getBobot() {
        return lurus + (belok_kanan*2);
    }
}
```

Lalu buat kelas Jalur *Comparator* untuk membandingkan jalur berdasarkan berat yang dihitung dengan rumus:
Bobot=(jumlah mobil lurus) + (jumlah mobil belok kanan)*2
Apabila ada jalur yang memiliki bobot sama, maka jalur tersebut akan diurutkan berdasarkan lastSelected terkecil. Kelas Comparator ini berfungsi saat membuat PriorityQueue dimana PriorityQueue tersebut akan membandingkan jalur berdasarkan kode yang sudah di override seperti dibawah ini:

```
public class JalurComparator implements
Comparator<Jalur> {
    @Override
    public int compare(Jalur j1, Jalur j2) {
        int bobotComparison =
Integer.compare(j2.getBobot(),
j1.getBobot());
        if (bobotComparison == 0) {
            return
Integer.compare(j1.lastSelected,
j2.lastSelected);
        }
        return bobotComparison;
    }
}
```

Buat main *class* nya dengan menginisiasi 4 jalur awal, lalu membuat *PriorityQueue* untuk memilih jalur mana yang akan diberikan warna hijau, Map untuk mendapatkan jalur dengan mudah berdasarkan ide, dan *ArrayList* untuk menyimpan hasil jawaban. Lalu Buat inisialisasi ke empat antrean jalur pada *PriorityQueue* dengan mengisikan kapasitas awal jalur.

```
// Inisiasi 4 jalur
int[][] jumlahKendaraanPerJalur = {
        {50, 100},
        {40, 40},
        {30, 0},
        {0, 30}
};

PriorityQueue<Jalur> queue = new
PriorityQueue<>(new JalurComparator());
Map<Integer, Jalur> jalurMap = new
HashMap<>();
ArrayList<Jalur> jalurList = new
ArrayList<>();

// Inisialisasi antrian jalur
for (int i = 0; i <
jumlahKendaraanPerJalur.length; i++) {
    Jalur jalur = new Jalur(i,
jumlahKendaraanPerJalur[i][0],
jumlahKendaraanPerJalur[i][1]);
    queue.add(jalur);
    jalurMap.put(i, jalur);
}
```

Lakukan iterasi sampai *Queue empty*. Ada banyak faktor untuk menyempurnakan iterasi ini yaitu:

1. Jika dalam 6 iterasi jalur tidak terpilih, pilih jalur itu, jika belum ada yang lebih dari 6, maka pilih jalur dengan melakukan pop pada *PriorityQueue* dan masukan hasilnya di *ArrayList*

```
//jika 6 iterasi belum dipilih, maka pilih
jalur itu
for (Jalur jalur : queue) {
    if (iterasi - jalur.lastSelected - 1 >=
6) {
        selectedJalur = jalur;
        break;
    }
}
if (selectedJalur == null) {
    selectedJalur = queue.poll();
} else {
    queue.remove(selectedJalur);
}
jalurList.add(selectedJalur);
```

2. Hapus jalur pada *queue* dan kurangi kapasitas (lurus - 20, belok_kanan - 10). Jika kapasitas tidak 0 baik jalur maupun belok_kanan, maka masukan lagi jalur ke dalam *PriorityQueue*.

```
// Kurangi jumlah mobil di jalur tersebut
selectedJalur.belok_kanan = Math.max(0,
selectedJalur.belok_kanan - 10);
selectedJalur.lurus = Math.max(0,
selectedJalur.lurus - 20);
selectedJalur.lastSelected = iterasi;

// Jika masih ada mobil, masukkan kembali ke
dalam antrian
if (selectedJalur.belok_kanan > 0 ||
selectedJalur.lurus > 0) {
    queue.add(selectedJalur);
}
```

3. Setiap 4 iterasi, pengguna akan diminta untuk memasukkan tambahan kapasitas mobil pada tiap jalur untuk mensimulasikan keadaan nyata yang dimana jalur pasti mengalami penambahan antrean. Disinilah

keuntungan adanya algoritma *greedy* dibanding yang lain.

```
// Setiap 4 iterasi, meminta input pengguna
if (iterasi % 4 == 0) {
    System.out.println("Ingin menambahkan
kapasitas mobil?");
    String pilihan = scanner.nextLine();
    if(pilihan == "y"){
        System.out.println("Jumlah mobil
setiap jalur:");
        for (int i = 0; i <
jumlahKendaraanPerJalur.length; i++)
        {
            int tambah_belokKanan =
scanner.nextInt();
            int tambah_lurus =
scanner.nextInt();

            Jalur jalur = jalurMap.get(i);
            if (queue.contains(jalur)) {
                queue.remove(jalur);
                jalur.belok_kanan +=
tambah_belokKanan;
                jalur.lurus += tambah_lurus;
                queue.add(jalur);
            } else if (selectedJalur.id == i)
{
                selectedJalur.belok_kanan +=
tambah_belokKanan;
                selectedJalur.lurus +=
tambah_lurus;
            } else {
                queue.add(new Jalur(i,
tambah_belokKanan, tambah_lurus));
            }
        }
    }

}
```

4. Jika *PriorityQueue* telah habis, maka *print* hasilnya

```
System.out.print("Hasil = [");
for (int i = 0; i < jalurList.size(); i++) {
    System.out.print(jalurList.get(i).id +
1);
    if (i != jalurList.size() - 1) {
        System.out.print(", ");
    }
}
System.out.print("]");
```

V. CONTOH KASUS

Misalkan di suatu persimpangan memiliki kepadatan antrean seperti berikut
Jalur 1: 50 belok_kanan, 100 lurus
Jalur 2: 40 belok_kanan, 40 lurus
Jalur 3: 30 belok_kanan, 0 lurus
Jalur 4: 0 belok_kanan, 30 lurus

Buat PriorityQueue PQ lalu menginisiasikannya dengan 4 jalur yang sudah ada:
PQ [1(200), 2(120), 3(60), 4(30)]
Mulai Proses 4 iterasi pertama

- Iterasi 1:
  Ambil jalur 1, sehingga kapasitasnya menjadi 40 belok_kanan dan 80 lurus. Lalu perbarui PQ
  PQ [1(160), 2(120), 3(60), 4(30)]
- Iterasi 2:
  Ambil jalur 1, sehingga kapasitasnya menjadi 30 belok_kanan dan 60 lurus. Lalu perbarui PQ
  PQ [2(120), 1(120), 3(60), 4(30)]
- Iterasi 3:
  Ambil jalur 2, sehingga kapasitasnya menjadi 30 belok_kanan dan 20 lurus. Lalu perbarui PQ
  PQ [1(120), 2(80), 3(60), 4(30)]
- Iterasi 4:
  Ambil jalur 1, sehingga kapasitasnya menjadi 20 belok_kanan dan 40 lurus. Lalu perbarui PQ
  PQ [2(80), 1(80), 3(60), 4(30)]

Untuk mensimulasikan data asli, data pada jalur 4 diperbarui dengan menambahkan 25 Mobil Belok Kanan dan 50 mobil lurus sehingga menjadi seperti ini:
Jalur 1: 20 belok_kanan, 40 lurus
Jalur 2: 30 belok_kanan, 20 lurus
Jalur 3: 30 belok_kanan, 0 lurus
Jalur 4: 25 belok_kanan, 80 lurus
Dan PQ [4(130), 2(80), 1(80), 3(60)]

Lalu lanjutkan iterasi sampai iterasi yang ke 8

- Iterasi 5:
  Ambil jalur 4, sehingga kapasitasnya menjadi 15 belok_kanan dan 60 lurus. Lalu perbarui PQ
  PQ [4(90), 2(80), 1(80), 3(60)]
- Iterasi 6:
  Ambil jalur 4, sehingga kapasitasnya menjadi 5 belok_kanan dan 40 lurus. Lalu perbarui PQ
  PQ [2(80), 1(80), 3(60), 4(50)]
- Iterasi 7:
  Iterasi ini punya perilaku khusus karena jalur 3 tidak diberi lampu hijau selema 6 iterasi. Maka untuk Langkah ke 7 ini ambil jalur 3 , sehingga kapasitasnya menjadi 20 belok_kanan dan 0 lurus. Jika hasil pengurangan negative, maka set ke 0. Lalu perbarui PQ
  PQ [2(80), 1(80), 4(50), 3(40)]
- Iterasi 8:
  Ambil jalur 2, sehingga kapasitasnya menjadi 20 belok_kanan dan 0 lurus. Lalu perbarui PQ
  PQ [1(80), 4(50), 3(40), 2(40)]

Untuk mensimulasikan data asli, data pada jalur 3 diperbarui dengan menambahkan 20 mobil lurus sehingga menjadi seperti ini:
Jalur 1: 20 belok_kanan, 40 lurus
Jalur 2: 20 belok_kanan, 0 lurus
Jalur 3: 20 belok_kanan, 20 lurus
Jalur 4: 5 belok_kanan, 40 lurus
Dan PQ [1(80), 3(60), 4(50), 2(40)]

Lalu lanjutkan iterasi sampai selesai dan anggap tidak ada lagi penambahan jalur untuk mempersingkat contoh simulasi.

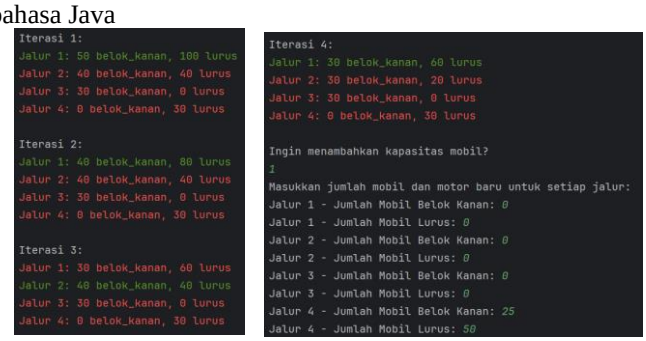
- Iterasi 9:
  Ambil jalur 1, sehingga kapasitasnya menjadi 10 belok_kanan dan 20 lurus. Lalu perbarui PQ
  PQ [3(60), 4(50), 2(40), 1(40)]
- Iterasi 10:
  Ambil jalur 3, sehingga kapasitasnya menjadi 10 belok_kanan dan 0 lurus. Lalu perbarui PQ
  PQ [4(50), 2(40), 1(40), 3(20)]
- Iterasi 11:
  Ambil jalur 4, sehingga kapasitasnya menjadi 0 belok_kanan dan 20 lurus. Lalu perbarui PQ
  PQ [2(40), 1(40), 3(20), 4(20)]
- Iterasi 12:
  Ambil jalur 2, sehingga kapasitasnya menjadi 10 belok_kanan dan 0 lurus. Lalu perbarui PQ
  PQ [1(40), 3(20), 4(20), 2(20)]

- Iterasi 13:
  Ambil jalur 1, sehingga kapasitasnya menjadi 0 belok_kanan dan 0 lurus. Lalu perbarui PQ dengan menghapus jalur 1 karena sudah 0.
  PQ [3(20), 4(20), 2(20)]
- Iterasi 14:
  Ambil jalur 3, sehingga kapasitasnya menjadi 0 belok_kanan dan 0 lurus. Lalu perbarui PQ dengan menghapus jalur 3 karena sudah 0.
  PQ [4(20), 2(20)
- Iterasi 15:
  Ambil jalur 4, sehingga kapasitasnya menjadi 0 belok_kanan dan 0 lurus. Lalu perbarui PQ dengan menghapus jalur 4 karena sudah 0.
  PQ [2(20)]
- Iterasi 16:
  Ambil jalur 2, sehingga kapasitasnya menjadi 0 belok_kanan dan 0 lurus. Lalu perbarui PQ dengan menghapus jalur 2 karena sudah 0.
  PQ []

Hasil akhirnya yaitu:

Hasil = [1, 1, 2, 1, 4, 4, 3, 2, 1, 3, 4, 2, 1, 3, 4, 2,2]
jalur 1 diberi lampu hijau selama 2 menit
jalur 2 diberi lampu hijau selama 1 menit
jalur 1 diberi lampu hijau selama 1 menit
Ada penambahan pada jalur 4 sebanyak 25 Mobil Belok Kanan dan 50 mobil lurus
Jalur 4 diberi lampu hijau selama 2 menit
Jalur 3 diberi lampu hijau selama 1 menit
Jalur 2 diberi lampu hijau selama 1 menit
Ada penambahan pada jalur 3 sebanyak 20 mobil lurus
Jalur 1 diberi lampu hijau selama 1 menit
Jalur 3 diberi lampu hijau selama 1 menit
Jalur 4 diberi lampu hijau selama 1 menit
Jalur 2 diberi lampu hijau selama 1 menit
Jalur 1 diberi lampu hijau selama 1 menit
Jalur 3 diberi lampu hijau selama 1 menit
Jalur 4 diberi lampu hijau selama 1 menit
Jalur 2 diberi lampu hijau selama 1 menit

Berikut ini hasil dari implementasi algoritma menggunakan bahasa Java

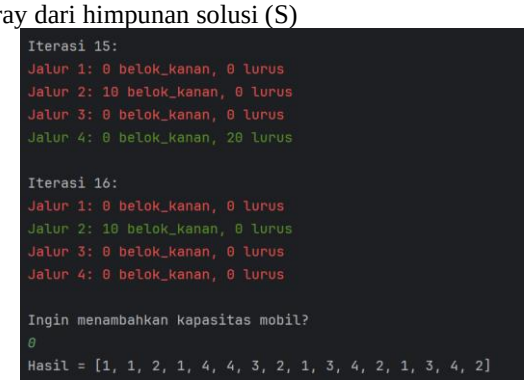


Gambar 5. Hasil Algoritma
(Sumber : Arsip Pribadi)

Warna hijau menandakan bahwa jalur tersebut sedang diberi lampu hijau / jalan. Dengan catatan proses pengurangan kapasitas tidak ditampilkan pada layer sehingga deskripsi jalur jalur tersebut merupakan kapasitas saat ini sebelum dikurangkan.

Tiap 4 iterasi, user akan dapat memperbarui data dengan input 1 jika ingin menambah atau input 0 jika tidak ingin menambah. Saat pengguna memasukan input 1, maka program akan meminta pengguna untuk memasukan tambahan kapasitas mobil untuk tiap jalur baik mobil yang akan jalan lurus ataupun mobil yang akan belok kanan.

Pada iterasi akhir, algoritma akan memunculkan hasil berupa listArray dari himpunan solusi (S)



Gambar 6. Hasil Algoritma
(Sumber : Arsip Pribadi)

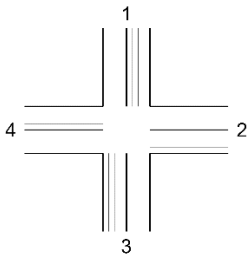
VI. PENGEMBANGAN ALGORITMA

Namun Pendekatan Greedy seperti penjelasan sebelumnya masih memiliki banyak kekurangan. Yaitu hanya dapat menjalankan 1 jalur saja. Algoritma ini dapat ditingkatkan dengan mengubah elemen elemen penyusun algoritma greedynya. Berikut ini pengubahan algoritma dengan upaya untuk mendapat hasil yang lebih efisien lagi.

A. Identifikasi Elemen Greedy

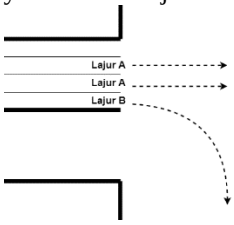
Sistem lampu lalu lintas otomatis dapat diidentifikasikan menurut elemen-elemen pada algoritma *greedy* sebagai berikut:

1. Himpunan Kandidat (C)
   Himpunan kandidat pada persoalan ini yaitu kombinasi dari 2 lajur. Misalkan ada 4 jalur yaitu jalur 1,2,3, dan 4 seperti di bawah ini



Gambar 7. Persimpangan 4 jalur
(Sumber : Arsip Pribadi)

Tiap jalurnya mempunyai 3 lajur, anggap lajur paling kiri untuk belok kiri dan tidak perlu mengikuti lampu sehingga kita hanya melihat 3 lajur ini saja.



Gambar 8. Lajur di suatu jalur
(Sumber : Arsip Pribadi)

Anggaplah lajur A untuk mobil yang akan lurus, dan lajur B untuk mobil yang akan belok kanan. Sehingga kandidat lajurnya yaitu 1A, 1B, 2A, 2B, 3A, 3B, 4A, 4B. Dengan angka menandakan jalur dan huruf menandakan lajur. Misal 1A, berarti itu merupakan kode untuk jalur 1 dengan lajur A.

Lajur yang dikombinasikan nantinya akan mendapatkan lampu hijau secara bersamaan sehingga untuk mencari kombinasi dari kedua jalur tidak boleh asal. Harus memperhatikan juga keselamatan pengguna seperti lajur yang dikombinasikan tidak boleh bertabrakan.

Kombinasi jalur yang aman dan tidak bertabrakan yaitu kombinasi lajur yang ada di jalur yang sama

{1A, 1B}
{2A, 2B}
{3A, 3B}
{4A, 4B}

Serta kombinasi lajur lurus yang saling berseberangan

{1A, 3A}
{2A, 4A}

Maka dari itu, himpunan kandidat C-nya

C = [{1A, 1B}, {2A, 2B}, {3A, 3B}, {4A, 4B}, {1A, 3A}, {2A, 4A}]

2. Himpunan Solusi (S)

   Himpunan solusi pada masalah ini yaitu urutan dari kombinasi lajur yang dihasilkan untuk mengurai kemacetan.

3. Fungsi Solusi

   Dikarenakan persoalan ini tidak punya solusi terbaik yang mutlak, fungsi solusi pada persoalan ini dicek pada tiap langkahnya apakah telah memilih langkah yang paling besar.

4. Fungsi Seleksi

   Fungsi seleksi pada persoalan ini ditentukan dengan menghitung bobot kombinasi lajur terbesar.

   Setelah jalur dipilih, kurangi kapasitas mobil pada lajur dan masukan kembali ke himpunan kandidat. Dengan ketentuan pengurangan 20 mobil yang akan lurus dan pengurangan 10 mobil yang belok kanan.

5. Fungsi Kelayakan

   Banyak sekali fungsi kelayakan yang harus didefinisikan pada persoalan ini untuk meningkatkan efisiensi.

   Jika suatu jalur tidak diberi lampu hijau selama 6 iterasi, maka untuk langkah selanjutnya yang akan dipilih adalah jalur tersebut.

   Untuk menstimulasikan data real dimana mobil akan terus berdatangan, maka dibuat proses masukan *input* pengguna tiap 4 iterasi untuk *update* tambahan mobil yang masuk ke jalur.

6. Fungsi Obyektif

   Fungsi Obyektif dalam persoalan ini adalah memaksimalkan bobot yang ada pada jalur dengan mempertimbangkan kapasitas mobil yang akan lurus dengan mobil yang akan belok kanan.

Berikut merupakan penyelesaian dari contoh kasus yang sama seperti sebelunya dengan

Jalur 1: 50 belok_kanan, 100 lurus
Jalur 2: 40 belok_kanan, 40 lurus
Jalur 3: 30 belok_kanan, 0 lurus
Jalur 4: 0 belok_kanan, 30 lurus

Iterasi	1A, 1B	2A, 2B	3A, 3B	4A, 4B	1A, 3A	2A, 4A	Jalur Pilihan
1	100,50 (200)	40,40 (120)	0,30 (60)	30,0 (30)	100,0 (100)	40,30 (70)	1A, 1B
2	80,40 (160)	40,40 (120)	0,30 (60)	30,0 (30)	80,0 (80)	40,30 (70)	1A, 1B
3	60,30 (120)	40,40 (120)	0,30 (60)	30,0 (30)	60,0 (60)	40,30 (70)	2A, 2B
4	60,30 (120)	20,30 (80)	0,30 (60)	30,0 (30)	60,0 (60)	20,30 (50)	1A, 1B
4’	40,20 (80)	20,30 (80)	0,30 (60)	80,25 (130)	40,0 (40)	20,80 (100)	Penambahan
5	40,20 (80)	20,30 (80)	0,30 (60)	80,25 (130)	40,0 (40)	20,80 (100)	4A, 4B
6	40,20 (80)	20,30 (80)	0,30 (60)	60,15 (90)	40,0 (40)	20,60 (580)	4A, 4B
7	40,20 (80)	20,30 (80)	0,30 (40)	40,5 (50)	40,0 (40)	20,40 (60)	3A, 3B
8	40,20 (80)	20,30 (80)	0,20 (40)	40,5 (50)	40,0 (40)	20,40 (60)	2A, 2B
8’	40,20 (80)	0,20 (40)	20,20 (60)	40,5 (50)	40,20 (60)	0,40 (40)	Penambahan
9	40,20 (80)	0,20 (40)	20,20 (60)	40,5 (50)	40,20 (60)	0,40 (40)	1A, 1B
10	20,10 (40)	0,20 (40)	20,20 (60)	40,5 (50)	20,20 (40)	0,40 (40)	3A, 3B
11	20,10 (40)	0,20 (40)	0,10 (20)	40,5 (50)	20,0 (20)	0,40 (40)	4A, 4B
12	20,10 (40)	0,20 (40)	0,10 (20)	20,0 (20)	20,0 (20)	0,20 (20)	2A, 2B
13	20,10 (40)	0,10 (20)	0,10 (20)	20,0 (20)	20,0 (20)	0,20 (20)	1A, 1B
14	0,0 (0)	0,10 (20)	0,10 (20)	20,0 (20)	0,0 (0)	0,20 (20)	2A, 4A
15	0,0 (0)	0,10 (20)	0,10 (20)	0,0 (0)	0,0 (0)	0,0 (0)	3A, 3B
16	0,0 (0)	0,10 (20)	0,0 (0)	0,0 (0)	0,0 (0)	0,0 (0)	2A, 2B
-	0,0 (0)	0,0 (0)	0,0 (0)	0,0 (0)	0,0 (0)	0,0 (0)	-

Tabel 1. Proses mendapatkan hasil

Keterangan: 100,50 (200) Berarti Kapasitas mobil lurus ada 100 dan kapasitas mobil belok_kanan ada 50. Angka 200 menunjukan bobot dari jalur tersebut.

Ternyata tidak ada perubahan yang signifikan setelah mengubah himpunan kandidat menjadi kombinasi dari 2 lajur yang tidak bertabrakan. Iterasi atau waktu yang dibutuhkan tetaplah 16 iterasi / 16 menit.

Ada perbedaan pada pengambilan di iterasi 14,15,16.Namun hal ini tidak mengubah hasil dan tetap memberikan jalur ke lajur yang sama dengan algoritma sebelum ini (himpunan kandidatnya berdasarkan jalur).

VII. Link Video Youtube

Penulis juga membuat video yang menjelaskan penerapan algoritma *greedy* pada sistem lampu lalu lintas otomatis pada *link* berikut: https://youtu.be/s29N2B6ILnQ

VIII. Link GitHub

Kode lengkap implementasi algoritma *greedy* dapat diakses melalui link berikut:
https://github.com/keanugonza/Makalah-Strategi-Algoritma.git

IX. KESIMPULAN

Algoritma Greedy sangat cocok digunakan dalam persoalan ini karena persoalan ini memungkinkan untuk menambah / memperbarui data jalur yang ada. Jika menggunakan algoritma non-Greedy misalnya A*, maka tidak memungkinkan untuk adanya penggantian data didalam-nya.

Untuk menyempurnakan hasil, ada beberapa batasan batasan yang perlu diterapkan yaitu, bila ada jalur yang tidak terpilih selama 6 iterasi, maka di iterasi berikutnya jalur tersebut yang akan dipilih. Hal ini dilakukan agar tidak membuat mobil menunggu terlalu lama di persimpangan.

Penulis menerapkan 2 pendekatan Greedy dengan himpunan kandidat yang berbeda. Algoritma pertama menggunakan himpunan kandidat Jalur 1, 2, 3,dan 4 sedangkan algoritma kedua menggunakan himpunan kandidat kombinasi antara 2 jalur yang tidak bertabrakan, yaitu [{1A, 1B}, {2A, 2B}, {3A, 3B}, {4A, 4B}, {1A, 3A}, {2A, 4A}].

Dari 2 algoritma tersebut, ternyata tidak mengalami banyak perbedaan bahkan hampir dibilang sama persis. Hal ini dikarenakan himpunan kandidat kombinasi sebenarnya adalah jalur 1,2,3,4 ditambah 1A, 3A dan 2A, 4A. Apabila jalur 1 diberi lampu hijau, maka bobot pada jalur 1 itu sendiri berkurang serta pada 1A, 3A juga berkurang. Sama-sama berkurangnya bobot lajur ini membuat kombinasi 1A, 3A dan 2A, 4A susah untuk dipilih berdasarkan fungsi solusi.

Pada skenario normal seperti pada contoh kasus, kedua algoritma memerlukan waktu 16 menit untuk mengurai kemacetan yang sama dengan adanya tambahan kapasitas pada iterasi ke 4 dan juga 8.

Algoritma Greedy ternyata dapat dipakai ke sebuah persoalan yang solusi terbaik mutlaknya tidak perlu diperhatikan seperti pada kasus persoalan sistem lampu lalu lintas otomatis ini. Algoritma Greedy berfungsi untuk menyelesaikan persoalan ini dengan memberikan solusi hampirannya.

Dari sini, kita dapat mengetahui bahwa Algoritma Greedy memiliki kelebihan seperti kecepatan dalam mengeksekusi program karena kesederhanaannya, bisa mendapatkan solusi hampiran yang cukup baik

Namun disisi lain, algoritma Greedy juga punya kelemahan. Kelemahannya yaitu hasil yang didapatkan tidak selalu menghasilkan hasil yang optimal misal dengan bobot yang sama yaitu 40. Jalur 1 punya 20 mobil lurus dan 10 mobil belok kanan, sedangkan jalur 2 punya 40 mobil lurus. Jika menurut greedy mereka ini sama-sama saja, namun ternyata jalur 1 akan lebih menghasilkan keuntungan / efisiensi yang baik karena jalur 1 akan mengurangi total 30 mobil (20 mobil lurus dan 10 mobil belok kanan) sedangkan jalur 2 hanya mengurangi 20 mobil lurus.

X. UCAPAN TERIMA KASIH

Penulis mengucapkan puji syukur kepada Tuhan Yang Maha Esa atas segala Rahmat dan karunia yang diberikan-Nya sehingga penulis dapat menyelesaikan makalah berjudul “Analisis Penerapan Algoritma Greedy pada Sistem Lampu Lalu Lintas Otomatis” dengan baik. Penulis juga mengucapkan terima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M. T., Ibu Dr. Nur Ulfa Maulidevi, S.T., M.Sc, dan Bapak Ir.Rila Mandala, M.Eng.,Ph.D., yang telah membimbing dan mengajar mata kuliah IF2211 Strategi Algoritma selama satu semester ini. Terima kasih juga kepada pihak-pihak yang telah membantu penulis dalam menyelesaikan makalah ini.

REFERENSI

[1] Munir, Rinaldi. Algoritma Greedy (Bagian 1). Bandung: Informatika, 2021. Diakses pada 11 Juni, 2024 pukul 20.00. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Greedy-(2021)-Bag1.pdf

[2] Geeks For Geeks, Greedy Algorithms. Diakses 12 Juni, 2024 pukul 08.00. https://www.geeksforgeeks.org/greedy-algorithms/

[3] Alabi, Tantoluwa Heritage. 2023. What is a Greedy Algorithm? Examples of Greedy Algorithms. Diakses 12 Juni, 2024 pukul 08.00. https://www.freecodecamp.org/news/greedy-algorithms/

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Keanu Amadius Gonza Wrahatno (13522082)