Penerapan Algoritma A dalam Perancangan Jalur Kereta Api*

I Kadek Yuda Budipratama Giri / 13516115
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
yudaikadek22@gmail.com

***Abstrak*—Kereta api merupakan salah satu alat transportasi massal yang berperan penting dalam kehidupan manusia, karena banyak orang mengandalkan kereta untuk berpergian dan mengantarkan barang-barang seperti komoditas, batu bara, dan lain-lain dalam waktu yang singkat. Karena kecepatan kereta yang relatif tetap, jarak yang ditempuh oleh kereta dalam bepergian harus minimal untuk mencapai tujuan dengan waktu terdekat. Selain itu, perancangan yang buruk dapat menyebabkan biaya pembuatan jalur kereta yang tinggi, sehingga dapat menyebabkan kenaikan biaya pembuatan sistem kereta. Untuk mengurangi biaya pembuatan dan mempersingkat waktu tempuh, dibutuhkan rancangan jalur kereta yang menempuh jarak terpendek dari satu tujuan ke tujuan lain. Pada makalah ini, akan dijelaskan bagaimana algoritma A* (A-star) dapat memyelesaikan masalah mengenai perancangan jalur kereta terpendek untuk setiap tujuan dari setiap tempat.**

Keywords— Kereta Api, Perancangan Jalur, Jarak Terpendek, Algoritma A*, Pathfinding

I. PENDAHULUAN

Kereta api adalah alat transportasi yang pada saat ini menjadi pilihan banyak orang dalam bepergian. Orang banyak memanfaatkan kereta untuk pergi bekerja, pergi wisata, berpindah ke tempat yang jauh, mengantar barang, dan hal-hal lainnya. Bahkan, beberapa orang sudah tergantung dengan kereta karena tidak ada lagi kendaraan atau transportasi lain yang dapat digunakan selain kereta. Selain itu, harganya yang cukup murah, jarak tempuh yang cepat, dan hampir tidak ada hambatan juga menjadi keunggulan kereta dibandingkan alat transportasi darat seperti mobil atau motor.

Untuk mempertahankan keunggulan kereta dibandingkan transportasi lain, pelayanan kereta harus dibuat agar lebih efektif dan efisien. Hal-hal yang mempengaruhi kualitas pelayanan kereta adalah ketepatan waktu, keamanan, dan seberapa cepat waktu yang dibutuhkan untuk mencapai suatu tujuan.

Ketepatan waktu dapat diperoleh dari pengaturan jadwal kereta dan keberadaan kereta untuk mengurangi adanya sistem menunggu kereta yang berhenti untuk memberi jalan kereta lain yang kebetulan ada di rel yang sama dengan arah yang berbeda.

Keamanan dapat ditingkatkan dengan menjamin keselamatan setiap penumpang dalam perjalanan dengan kereta, seperti memastikan kualitas rel yang dilalui untuk mencegah kereta anjlok, memastikan keamanan stasiun dari kejahatan dan bencana, dan berbagai hal lainnya yang menyangkut keselamatan penumpang.

Waktu yang dibutuhkan untuk mencapat suatu tujuan harus seminimal mungkin, karena orang memilih kereta sebagai sarana transportasi untuk mengurangi waktu di perjalanan. Dengan tuntutan waktu seminimal mungkin dan kereta yang kecepatannya konstan, salah satu cara untuk mempersingkat waktu perjalanan adalah dengan memperpendek jarak yang harus ditempuh kereta dalam bepergian. Jarak yang ditempuh oleh kereta dapat dikurangi dengan memperkecil jumlah rel yang dibutuhkan untuk mencapai tempat tersebut. Dengan kata lain, dengan mengurangi panjang jalan yang harus ditempuh oleh kereta api, kita dapat membuat perjalanan kereta lebih efektif. Pada pembuatan sistem kereta api, di sinilah proses perancangan menjadi penting. Untuk merancang jalur kereta api yang efektif, kita dapat menggunakan algoritma A*.

Algoritma ini dipakai dalam permasalahan ini karena permasalahan jalur kereta ini merupakan permasalahan perencanaan jalur terpendek yang ditempuh dengan adanya halangan di antara dua tempat tersebut. Halangan ini berupa jalur yang tidak dapat ditempuh oleh kereta, seperti danau besar, gunung tinggi, tanah yang tidak boleh dilewati, dan sebagainya. Selain itu, tingkat akurasi solusi yang diberikan oleh algoritma A* cukup tinggi, tergantung bagaimana kita memperkirakan perhitungan dengan algoritma ini.

II. DASAR TEORI

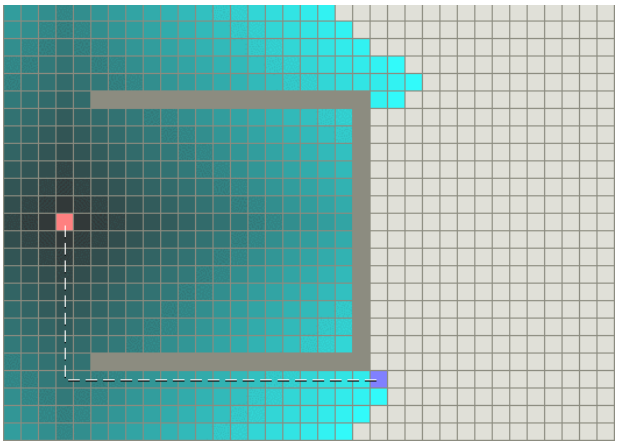
A. *Algoritma A**

Algoritma A* adalah salah satu algoritma yang sering digunakan untuk melakukan penelusuran graf maupun pencarian jalur (*pathfinding*)[3]. Algoritma ini digunakan untiik mengaproksimasi jalur dengan jarak terpendek dari tempat asal menuju tempat tujuan di mana terdapat halangan pada jalan tersebut.

Algoritma ini mengembangkan teknik yang digunakan dalam pencarian jalur menggunakan Algoritma Djikstra dengan mengambil beberapa karakteristik *Greedy Breadth-First-Search* (*Greedy BFS*) [3]. Alasan mengapa hal ini dilakukan adalah ketika kita ingin mencari suatu jarak terpendek dengan adanya rintangan dalam perjalanan, algoritma Djikstra membutuhkan waktu dan sumber daya lebih untuk memeriksa semua kemungkinan jalur yang ada untuk mencari jarak yang terpendek antara titik asal dan tujuan meskipun pencariannya menyeluruh dan pasti mendapatkan jarak yang paling minimum dari tujuan asal[1]. Sementara itu, *Greedy BFS* hanya

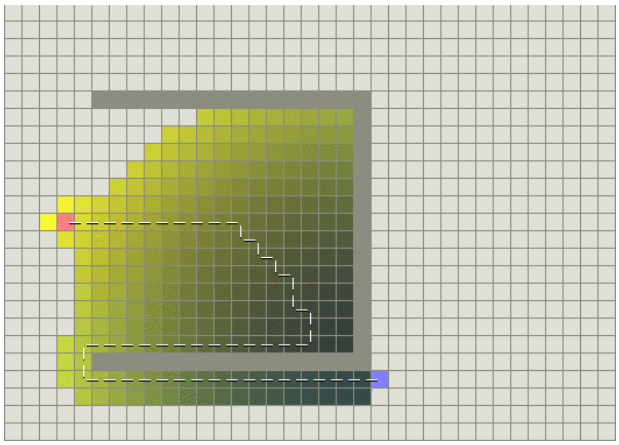
menelusuri titik yang memiliki jarak terpendek antara titik yang sedang dikunjungi dan tujuannya, akan tetapi pencarian seperti ini tidak selalu menghasilkan hasil yang optimal karena hanya mementingkan kandidat solusi yang sementara ini benar[1].

Misal terdapat sebuah peta yang direpresentasikan sebagai 2D *grid*, dengan satu kotak menggambarkan simpul dari graf permasalahan dan pergerakan dari satu kotak ke kotak lain menggambarkan sisi dari graf permasalahan. Dengan Algoritma Djikstra, setiap simpul diperiksa sampai mendapatkan simpul tujuan. Setiap simpul yang diperiksa ditandai warna hitam sampai biru muda yang menggambarkan jarak antara titik asal dan titik yang sedang dicari. Meskipun jalan yang didapat minimum, perhitungan yang dilakukan sangat banyak sehingga sangat memakan waktu.



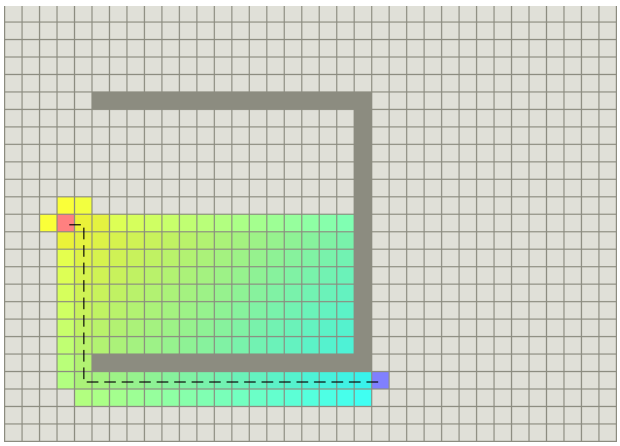
Gambar 1. Pencarian Jalur Terpendek dengan Algoritma Djikstra [1]

Sementara itu, algoritma *Greedy BFS* yang bekerja dengan mengikuti titik dengan jarak terpendek, saat mengerjakan hal yang sama, memiliki beban kerja yang lebih ringan dibandingkan algoritma Djikstra karena kandidat solusi telah dipotong berdasarkan sebuah fungsi heuristik berupa jarak antara titik yang dicari dengan titik tujuan dengan warna yang semakin kuning menandakan jarak antara titik asal dan titik tujuan. Akan tetapi, karena hanya berpegang pada kandidat solusi sementara tanpa memperhatikan adanya halangan, algoritma ini memberikan solusi yang tidak optimal seperti algoritma Djikstra.



Gambar 2. Pencarian Jalur Terpendek dengan Algoritma Greedy BFS [1]

Algoritma A* mengkombinasikan informasi yang digunakan oleh Algortima Djikstra berupa jarak antara titik asal dan titik yang sedang diproses dan algoritma Greedy BFS berupa jarak antara titik yang sedang diproses dengan titik tujuan. Dengan mencari nilai minimum dari fungsi heuristik tiap titik yang ditemukan pada setiap langkah, akan didapat jalur yang jaraknya tidak berbeda dengan algoritma Djikstra dengan waktu pemrosesan yang lebih cepat karena lebih sedikit titik yang harus dijelajahi.



Gambar 3. Pencarian Jalur Terpendek dengan Algoritma A*

Fungsi heuristik yang digunakan algoritma A* berasal dari penjumlahan antara nilai fungsi g(n) yang merepresentasikan biaya (*cost*) yang dibutuhkan untuk mencapai simpul yang sedang diproses dari simpul awal dan nilai fungsi h(n) yang merepresentasikan fungsi heuristik yang digunakan dalam memperkirakan biaya yang dibutuhkan untuk menemukan simpul tujuan dari simpul yang sedang diproses. Nilai dari fungsi heuristik f(n) untuk algoritma A* dapat dituliskan sebagai persamaan berikut

$$f(n) = g(n) + h(n) \quad (1)$$

Kemangkusan algoritma ini sangat bergantung kepada bagaimana fungsi heuristik yang digunakan. Oleh sebab itu,

fungsi heuristik harus ditentukan terlebih dahulu oleh pemogram sebelum melakukan perancangan algoritma.

B. Fungsi Heuristik

Fungsi heuristik adalah fungsi yang digunakan untuk mengevaluasi kandidat solusi yang dipilih dalam suatu proses algoritma A*. Fungsi ini akan mengatur bagaimana hasil dari pencarian dengan algoritma A*.

Merujuk pada persamaaan (1), kita dapat membuat pemrosesan A* menjadi lebih akurat atau lebih cepat. Jika nilai h(n) adalah 0, maka persamaan (1) dapat ditulis sebagai

$$f(n) = g(n)$$

dengan nilai fungsi g(n) adalah *cost* terkecil yang dibutuhkan untuk mencapai titik yang sedang diproses. Akibatnya, Algoritma A* akan berperan seperti algoritma Djikstra, yang akan menjamin bahwa jalur yang kita tentukan pasti efektif dengan usaha yang sangat besar[2].

Jika nilai h(n) menjadi sangat besar dibandingkan dengan nilai g(n), maka persamaan (1) dapat ditulis sebagai

$$f(n) = h(n)$$

dengan nilai h(n) adalah fungsi heuristik yang dipilih. Akibatnya algoritma A* akan berperan sebagai algoritma *Greedy BFS*, sehingga pemrosesan menjadi sangat cepat, tetapi dengan hasil yang tidak akurat[2]. Nilai h(n) yang lebih kecil akan membuat pencarian makin meluas, sementara h(n) yang besar dapat menyebabkan pemrosesan yang lebih cepat[2].

C. Kereta

Kereta api merupakan alat transportasi darat yang digunakan untuk mengangkut barang dan orang. Kereta api terdiri atas dua bagian, yaitu lokomotif yang menjadi pusat tenaga untuk menggerakan kereta, dan gerbong yang berisi muatan barang atau penumpang yang dibawa. Gerakan yang dihasilkan oleh kereta berasal dari lokomotif kereta yang berjalan. Kereta api menggunakan bahan bakar diesel dan dapat juga disebut kereta diesel.

Gambar 4. Kereta api dan rel kereta api sebagai jalurnya[4]

Kereta berjalan menggunakan jalur tersendiri yang disebut rel kereta. Rel kereta adalah jalur tersendiri yang digunakan kereta untuk bepergian dari suatu tempat ke tempat lain. Rel kereta berfungsi untuk menjaga kereta agar tidak keluar dari jalur tujuannya, karena kereta sendiri tidak punya alat kendali seperti mobil atau motor. Jika rel kereta tidak ada atau tidak dapat berfungsi dengan baik, kereta tidak dapat beroperasi dengan baik dan dapat menyebabkan kecelakaan. Rel dibangun sesuai dengan tipe kereta yang dilewati. Rel kereta juga dapat menandakan jalur mana saja yang dilalui kereta pada peta.

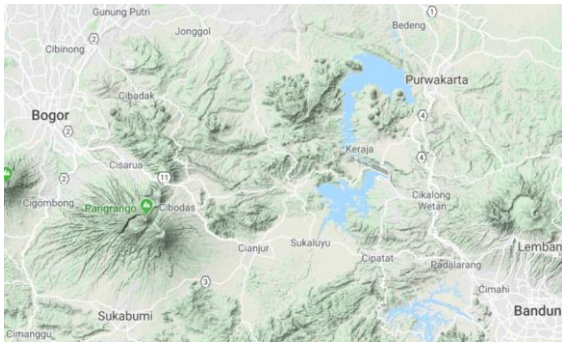
Gambar 5. Jalur kereta api pada peta [5]

Rel kereta api memiliki besi di tanah yang berfungsi menahan roda kereta api. Rel kereta api hanya bisa dirancang dan disusun satu kali karena rel terpasang pada tanah dan pemindahan rel kereta api dapat menyebabkan kerusakan pada rel. Selain itu, waktu dan biaya yang dibutuhkan juga sangat tinggi dan tidak sepadan dengan hasil yang ada, sehingga tidak efektif apabila rel berubah posisinya. Kereta bepergian dalam kecepatan yang relatif tetap sesuai dengan medan yang dilalui untuk menjaga kenyamanan penumpang dan menjaga agar barang bawaan yang dibawa tetap aman.

III. PERANCANGAN ALGORITMA

Untuk merancang algoritma untuk persoalan ini, ada tahapan yang harus dilakukan. Pertama, kita harus mengidentifikasi peta, medan yang boleh dilalui, dan medan yang tidak dapat dilalui. Setelah itu, kita harus menentukan fungsi heuristik yang cocok digunakan untuk memperkirakan jarak antara satu titik dengan titik yang lain untuk mengevaluasi kandidat titik yang menjadi solusi. Setelah fungsi heuristik ditentukan, maka barulah algoritma dapat kita susun untuk masalah ini.

Untuk menggambarkan permasalahan di atas, akan digunakan suatu peta daerah dengan dua kota, Kota Bogor dan Kota Purwakarta sebagai lokasi awal dan lokasi tujuan dari kereta kita.



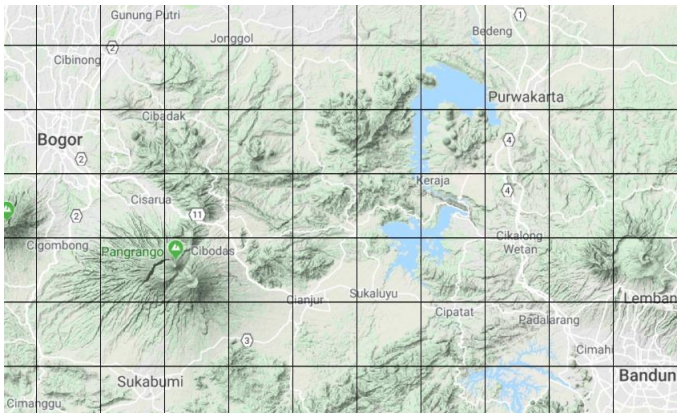
Gambar 6. Peta dari contoh permasalahan

Diambil dari https://www.google.co.id/maps/

A. *Memetakan Daerah Permasalahan*

Untuk menyelesaikan masalah mengenai perencanaan jalur kereta, terlebih dahulu dilakukan pemetaan daerah menjadi sebuah *grid* 2D. Alasan mengapa *grid* 2D digunakan untuk memetakan persoalan ini adalah untuk membagi tiap daerah menjadi kotak-kotak yang lebih mudah menggambarkan simpul-simpul dari graf permasalahan, sehingga penerapan algoritma A* dapat dilakukan dengan lebih mudah dan akurat.

Pemetaan ke dalam *grid* 2D dilakukan dengan memberikan garis-garis penanda pada peta secara membujur dan melintang dengan jarak yang sama, sehingga setiap bagian peta dapat terbagi menjadi wilayah kotak-kotak kecil. Apabila peta pada Gambar 6 kita representasikan dalam *grid 2D*, bentuknya akan seperti berikut.



Gambar 7. Representasi Peta dalam *Grid 2D*

Setelah peta diberikan *grid 2D*, kita dapat merujuk ke pada suatu lokasi pada peta seperti *array 2D*, di mana penomoran dimulai dari (0,0) pada posisi pojok kanan atas sampai pojok kanan bawah. Dengan demikian, lokasi kota bogor berada pada posisi (2,1), sementara kota Purwakarta berada pada posisi (1,8).

Selain menunjukkan lokasi kota asal dan tujuan, *grid 2D* juga dapat merepresentasikan lokasi dari daerah yang tidak mungkin dilalui oleh rel kereta. Pada permasalahan ini, diasumsikan kereta api tidak dapat melintasi danau dan gunung, karena melewati danau dan gunung membutuhkan usaha yang sangat sulit, sehingga kedua objek geografis ini akan dianggap sebagai rintangan (*obstacle*) yang berada di antara dua kota ini. Sungai tidak dijadikan *obstacle* pada permasalahan ini karena penyediaan rel yang melewati sungai masih mungkin dan masih bisa ditangani pembiayaannya.

Untuk mengidentifikasi adanya danau dan gunung pada suatu kotak, isi dari daerah kotak tersebut akan diperiksa. Jika isi dari daerah tersebut memiliki porsi lebih dari 50% berwarna biru, berarti lokasi tersebut mengandung danau atau daerah berair lain yang tidak dapat dilintasi kereta. Jika daerah tersebut berisi 50% berwarna hijau tua yang disertai dengan garis-garis lipatan gunung, maka daerah tersebut merupakan gunung. Contoh dari kotak yang merupakan danau ada pada (1,7), sementara contoh dari kotak yang mengandung gunung ada pada (0,3). Jika kotak tersebut tidak memenuhi syarat-syarat yang telah dijelaskan, maka kotak tersebut aman untuk dibuat jalur kereta. Untuk memudahkan penandaan jalur yang bisa dilalui dan tidak bisa dilalui, *grid* 2D dapat kita sederhanakan dengan *grid* 2D seperti Gambar 8, di mana S menandakan tempat kita mulai, F menandakan tujuan kita, dan X menandakan jalan yang tidak dapat dilewati rel kereta.

			X	X			X	F		
	S	X	X	X	X	X	X			
X			X	X	X	X	X		X	X
	X	X	X	X	X	X	X		X	X
	X						X			
			X	X	X	X	X	X		

Gambar 8. Representasi Peta dalam Grid 2D yang lebih sederhana

Untuk memudahkan pengolahan data pada komputer, *grid 2D* yang sudah dibuat akan kita ubah nenjadi sebuah matriks di mana jalur yang bisa dilewati, kota asal, dan kota tujuan memiliki nilai 1 dan jalur yang tidak bisa dilewati memiliki nilai 0. Matriks ini yang nanti akan menjadi representasi peta pada algoritma yang akan dibuat.

1	1	1	1	1	1	1	1	1	1	1
1	1	1	0	0	1	1	0	1	1	1
1	1	0	0	0	0	0	0	1	1	1
0	1	1	0	0	0	0	0	1	0	0
1	0	0	0	0	0	0	0	1	0	0
1	0	1	1	1	1	1	0	1	1	1
1	1	1	0	0	0	0	0	0	1	1

Gambar 9. Representasi Peta dalam Matriks

Perlu diperhatikan bahwa matriks ini bukan matriks bobot graf, tetapi matriks representasi peta. Tiap elemen dalam array merepresentasikan sebuah simpul dan jarak antara tiap kotak sama, sehingga tidak dibutuhkan lagi bobot tiap sisi antargraf.

B. *Memilih Fungsi Heuristik*

Ketika dibahas mengenai A*, diketahui bahwa fungsi heuristik sangat mempengaruhi hasil yang diperoleh dari algoritma A*. Untuk itu, fungsi heuristik yang tepat harus ditentukan. Fungsi heuristik h(n) yang baik idealnya memberikan kita nilai *cost* menuju tempat tujuan, tetapi hal tersebut tidak dimungkinkan terjadi karena program sendiri tidak tahu mengenai jalur mana yang terpendek sebelum ditelusuri[3]. Oleh sebab itu, kita akan memilih sebuah fungsi heuristik yang dapat memperkirakan nilai tersebut[3]. Oleh sebab itu, fungsi yang dipilih sebagai fungsi heuristik tidak boleh melebihi *cost* total yang dibutuhkan untuk mencapai tujuan akhir dari tujuan awal.

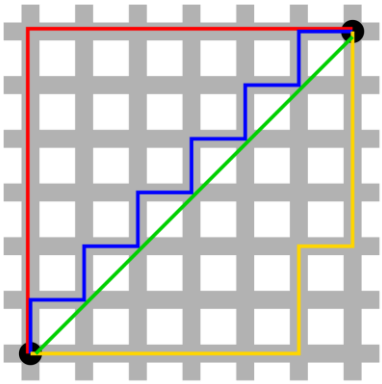
Ada dua fungsi heuristik yang dapat digunakan, yaitu *Manhattan Distance* dan *Euclidean Distance.*

1) Manhattan Distance

Manhattan Distance memperkirakan *cost* yang dibutuhkan dengan menghitung jumlah kotak yang harus dilalui secara horizontal maupun vertikal untuk mencapai tujuan akhir. Secara matematis, nilai Manhattan Distance untuk dua titik yang berbeda adalah

$$h = | x_{mulai} - x_{tujuan} | + | Y_{mulai} - Y_{tujuan} | \quad (2)$$

Heuristik ini akan memberikan hasil yang tepat apabila titik mulai dan titik tujuan memiliki nilai X atau Y yang sama, atau dengan kata lain, ketika jalur yang dipilih adalah jalur yang lurus[3] Manhattan Distance ditunjukkan dengan garis hijau muda.



Gambar 10. Jarak Manhattan dan Jarak Euclidean [3]

2) Euclidean Distance

Euclidean Distance adalah metode pencarian jarak antara dua titik. Biasanya heuristik ini digunakan jika arah gerak tidak dibatasi arahya, tetapi dalam kasus per *grid* pun bisa digunakan. Rumus dari Euclidean Distance adalah

$$h = \sqrt{(X_{mulai} - X_{tujuan})^2 + (Y_{mulai} - Y_{tujuan})^2} \quad (3)$$

Heuristik ini memberikan nilai yang lebih kecil daripada heuristik dengan Manhattan Distance, yang berarti heuristik ini memiliki tingkat akurasi yang tinggi. Akan tetapi, dengan tingkat akurasi yang makin tinggi dan nilai heuristik yang lebih kecil berarti waktu pencarian akan berlangsung lebih lama karena area yang dijelajahi semakin banyak. Euclidean Distance digambarkan oleh garis hijau pada Gambar 10.

Setelah diketahui dua fungsi heuristik tersebut, akan dipilih salah satu dari kedua fungsi heuristik tersebut untuk digunakan dalam persoalan ini. Untuk permasalahan perancangan jalur kereta yang tidak melakukan perpindahan kotak secara diagonal, dapat digunakan Manhattan Distance sebagai fungsi heuristik karena karakteristik dari Manhattan Distance yang hanya melakukan perpindahan dengan menggunakan perpindahan horizontal dan vertikal.

Sementara itu, Euclidean Distance biasanya dipakai dalam pencarian jalur AI (*artificial intelligence*) dalam permainan komputer yang membutuhkan pergerakan musuh karena memiliki pergerakan yang bebas. Hal itu bukan berarti kita tidak dapat menggunakan Euclidean Distance sebagai heuristik dalam permasalahan ini. Euclidean Distance memiliki keakuratan yang lebih tinggi dari Manhattan Distance dengan kekurangan memiliki waktu pemrosesan yang lebih lama.

Pada makalah ini, akan dipilih Euclidean Distance sebagai fungsi heuristik karena waktu proses pada kasus ini bukan faktor yang harus diperhatikan karena wilayah di dataran tidak akan berubah secara dinamis dalam waktu yang sangat cepat, sehingga perhitungan yang cepat tidak selalu dibutuhkan. Akan tetapi, keakuratan jalur yang paling pendek merupakan faktor yang penting dalam perancangan jalur kereta api, sehingga fungsi heuristik Euclidean Distance dipilih.

Pseudocode untuk menentukan nilai Euclidean distance adalah sebagai berikut

Pseudocode Euclidean Distance

```
function EuclideanDistance(start, finish) {
    dx = start.x - finish.x
    dy = start.y - finish.y
    return sqrt(dx*dx + dy*dy)
}
```

C. Algoritma Perancangan Jalur

Setelah kita merepresentasikan peta ke dalam bentuk yang dapat diolah komputer dan menentukan fungsi heuristik, sekarang akan dirancang algoritma A* untuk menentukan jalur yang dapat dilalui oleh kereta.

Perancangan jalur akan mengikuti algoritma A*, di mana dalam algoritma A* ini, kita akan menelusuri tiap kotak dari titik mulai sampai kita menemukan titik tujuan kita. Langkah-langkah yang harus dilakukan untuk melakukan pencarian adalah sebagai berikut

1. Buat suatu struktur data yang berisi koordinat lokasi suatu petak dan nilai f, g, dan h seperti persamaan (1) dari petak tersebut. Struktur data ini akan disebut sebagai “node”.
2. Buat sebuah list yang berisi kumpulan node. List ini berfungsi untuk menyimpan node yang telah ditemukan tetapi belum dievaluasi untuk kecocokan kandidat solusi. List ini diberi nama “open_list”
3. Buat sebuah list yang berisi kumpulan node. List ini berfungsi untuk menyimpan node yang telah dievaluasi oleh algoritma agar tidak diakses kembali oleh algoritma.
4. Buat sebuah map yang akan menyimpan pasangan antara suatu node dengan node lain, di mana salah satu node merupakan pendahulu dari node yang lain yang memiliki nilai f yang paling efektif. Kita akan beri nama map ini “cameFrom”. Map ini akan digunakan untuk menelusuri kembali jalur yang telah dilalui oleh jalur yang menjadi solusi.

5. Isi open_list dengan titik mulai dan nilai f, g, h dari titik mulai. nilai g diisi 0 (karena tidak ada jarak antara titik mulai dan titik mulai) dan h diisi dengan jarak heuristik antara titik mulai dan tujuan. Isi juga cameFrom[titik mulai] sebagai nilai kosong.
6. Selama open_list belum kosong atau tujuan akhir sudah ditemukan, lakukan evaluasi titik pada open_list dengan langkah sebagai berikut:
    a. Cari node dengan nilai f yang paling kecil dalam open list, missal kita namankan node tersebut "currNode".
    b. Keluarkan currNode dari open_list.
    c. Cari tetangga dari currNode. Karena currNode berupa petak dalam peta, tetangga dari currNode adalah petak di bagian utara, selatan, timur, dan barat dari currNode. Masukkan nilai currNode dengan *key* tiap tetangga pada cameFrom, sehingga nilai `cameFrom[tetangga] = currNode`
    d. Untuk setiap tetangga, periksa hal sebagai berikut
        i. Jika node tetangga tidak di luar batas peta dan bukan daerah yang tidak boleh dijalani, jangan proses node tetangga tersebut karena node tersebut tidak valid.
        ii. Jika node tersebut valid, apabila node tetangga tersebut merupakan tujuan akhir. Jika benar, maka pencarian dihentikan dan jalur dibuat dengan menelusuri kembali map cameFrom sampai bertemu dengan nilai kosong.
        iii. Jika node tersebut bukan tujuan akhir, hitung nilai f, g, dan h untuk node tersebut.
        iv. Jika ada node pada open list yang memiliki lokasi koordinat yang sama dengan node tetangga dan memiliki nilai f yang lebih kecil daripada node tetangga, lewati node tetangga tersebut.
        v. Jika ada node pada closed list yang memiliki lokasi koordinat yang sama dengan node tetangga dan memiliki nilai f yang lebih kecil daripada node tetangga, lewati node tetangga tersebut.
        vi. Jika kondisi pada poin iii dan iv tidak terpenuhi, maka masukkan node tetangga ke dalam open list.
    e. Masukkan currNode ke dalam closed list.
7. Jika sampai open list kosong belum sampai ke pada solusi, tampilkan pesan kesalahan bahwa titik tujuan tidak dapat tercapai.

Langkah-langkah di atas dapat dituliskan ke dalam sebuah pseudocode sebagai berikut

Pseudocode algoritma A*

```
function pathSeacrhAstar(start, goal) {
  // inisiasi nilai f, g, h pada start
  start.g = 0
  start.h = EuclideanDistance(start, goal)
  start.f = start.g + start.f
  // inisiasi open_list dan closed_list
  open_list = [start]
  closed_list = [ ]
  // inisiasi cameFrom
  cameFrom[start] = NULL
  // mulai penelusuran pada peta
  while open_list is not empty :
    currNode = node in open_list with minimum node.f
    open_list.remove(currNode)
    if (isValid(currNode)) :
      if (currNode.pos = goal.pos) :
        return reconstruct_path(cameFrom, currNode)
      else :
        for nodes in tetangga :
          nodes.g = currNode.g + 1
          nodes.h = EuclideanDistance(nodes, goal)
          nodes.f = nodes.g + nodes.f
          // cek apakah ada di closed list atau open list
          // jika idxClosed  == -1, nodes tidak ada pada
closed_list
          // jika idxOpen  == -1, nodes tidak ada pada
open_list
          idxClosed = searchIdx(closed_list, nodes)
              idxOpen = searchIdx(open_list, nodes)
```

```
          if (idxClosed != -1) and nodes.f >
closed_list[idxClosed].f :
            continue // lewati nodes yang ini
          else if (idxOpen != -1) and nodes.f >
open_list[idxOpen].f :
            continue // lewati nodes yang ini
          else :
            open_list.append(nodes)
          #end if
        #end for
      #end if
    #end if
    closed_list.add(currNode)
   # end while
   return error
}
```

D. Pengujian Algoritma

Apabila algoritma tersebut diimplementasikan dan contoh persoalan mengenai jalur kereta antara Bogor dan Purwakarta diselesaikan dengan algoritma di atas, akan muncul hasil sebagai berikut

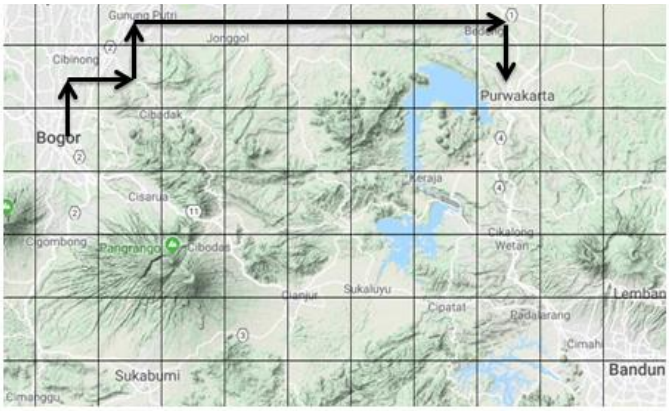
Hasil implementasi pada persoalan Kereta Bogor – Purwakarta

```
[1,1,1,1,1,1,1,1,1,1]
[1,1,1,0,0,1,1,0,1,1,1]
[1,1,0,0,0,0,0,0,1,1,1]
[0,1,1,0,0,0,0,0,1,0,0]
[1,0,0,0,0,0,0,0,1,0,0]
[1,0,1,1,1,1,1,0,1,1,1]
[1,1,1,0,0,0,0,0,0,1,1]

Lokasi Awal  : (2, 1)
Lokasi Tujuan: (1, 8)

Rute menuju tujuan telah ditemukan
Jalur yang terbentuk:
(2,1) -> (1,1) -> (1,2) -> (0,2) -> (0,3) -> (0,4) ->
(0,5) -> (0,6) -> (0,7) -> (0,8) -> (1,8)
```

Jika jalur yang terbentuk kita implementasikan ke dalam peta, maka jalur yang terbentuk adalah sebagai berikut



Gambar 11. Hasil Penyelesaian dengan Algoritma A*

Dapat dilihat meskipun terdapat rintangan di antara tempat mulai dan tujuannya, algoritma A* tetap dapat mencari rute dengan jarak terpendek tanpa menerobos gunung atau danau besar yang tidak dapat dilewati kereta. Selain itu, jarak yang ditempuh juga minimal, yaitu melewati 10 petak dihitung dari lokasi mulai.

IV. KESIMPULAN

Algoritma A* terbukti dapat menyelesaikan permasalahan perancangan jalur kereta dengan baik. Akan tetapi, kesuksesan dari algoritma A* sangat tergantung dari fungsi heuristik yang ditentukan. Jadi, untuk menyelesaikan perancangan jalur kereta, harus ditentukan pemetaan masalah dan heuristik yang digunakan sebelum memulai perancangan algoritmanya.

REFERENSI

[1] Introduction to A*
http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html
Diakses pada 12 Mei 2018

[2] Heuristics
http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html
Diakses pada 12 Mei 2018

[3] A* Search | Brilliant Math & Science Wiki
https://brilliant.org/wiki/a-star-search/
Diakses pada 12 Mei 2018

[4] Analisa Batas Tanah Penguasaan Kereta Api Indonesia
http://lampungreformaagraria.blogspot.co.id/2016/04/analisa-batas-tanah-penguasaan-kereta.html
Diakses pada 12 Mei 2018

[5] Malang with Malabar #1
https://yosnex.wordpress.com/2011/11/09/malang-with-malabar-1/
Diakses pada 12 Mei 2018

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, 13 Mei 2018

I Kadek Yuda B. G.
13516115