Penerapan Algoritma *Branch and Bound* untuk Menentukan Rute Distribusi Vaksin COVID-19 di Indonesia

Irvin Andryan Pratomo 13519162
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail : 13519162@std.stei.itb.ac.id

***Abstrak*—Saat ini vaksin COVID-19 sudah tersedia di Indonesia dan diharapkan dapat memberikan kekebalan untuk masyarakat agar terhindar dari COVID-19. Namun, dengan jumlahnya yang masih terbatas, rute distribusi vaksin harus optimal sehingga daerah dengan risiko penyebaran COVID-19 yang lebih tinggi dapat diprioritaskan. Salah satu cara yang dapat dilakukan untuk menentukan daerah yang akan mendapatkan vaksin terlebih dahulu adalah dengan memanfaatkan konsep *Travelling Salesman Problem* (TSP) yang dioptimasi menggunakan algoritma *Branch and Bound*. Pada artikel ini akan dibahas aplikasi algoritma *Branch and Bound* dalam menyelesaikan persoalan TSP untuk menentukan rute distribusi vaksin COVID-19.**

Kata kunci—vaksin COVID-19; rute; TSP; Branch and Bound

I. PENDAHULUAN

Coronavirus Disease 19 merupakan penyakit yang disebabkan oleh virus SARS-CoV-2. Virus ini ditemukan di Wuhan, *People Republic of China* pada tahun 2019 dan dilaporkan kepada WHO pada tanggal 31 Desember 2019 [1]. Penderita penyakit yang biasa disebut COVID-19 ini pada ummumnya menderita gejala seperti demam, batuk kering, dan kelelahan. Gejala lain yang lebih jarang tetapi mungkin ditemukan pada penderita COVID-19 diantaranya adalah kehilangan indra penciuman dan indra perasa, penyumbatan hidung, *conjunctivitis* atau radang konjungtiva, sakit tenggorokan, sakit kepala, sakit atau linu pada otot, ruam pada kulit, mual, diare, dan pusing. Pada kasus yang parah, penderita COVID-19 dapat merasa kesulitan untuk bernapas, kehilangan nafsu makan, sakit di bagian dada, dan temperatur tubuh yang tinggi.

Virus yang menyebabkan penyakit COVID-19 merupakan virus dengan jenis corona yang dapat dengan mudah berpindah inang. Salah satu cara penyebaran COVID-19 adalah melalui *droplet* seseorang yang menderita atau membawa virus Sars-CoV-2 tersebut [2]. Mudahnya penyebaran COVID-19 membuat banyak negara kewalahan menghadapinya, termasuk Indonesia. Salah satu cara agar pandemi COVID-19 dapat segera berakhir adalah dengan membangun kekebalan masyarakat terhadap virus Sars-CoV-2 tersebut melalui pemberian vaksin.

Gambar 1 Ilustrasi Vaksin COVID-19

(Sumber : https://www.halodoc.com/artikel/perlu-tahu-ini-fakta-lengkap-mengenai-vaksin-covid-19 diakses pada tanggal 8 Mei 2021)

Upaya pemerintah dalam melaksanakan vaksinasi di Indonesia sudah dimulai sejak bulan Januari 2021 [3] dengan target kecepatan satu juta vaksinasi setiap harinya dan target keseluruhan penduduk Indonesia yang perlu divaksinasi sebanyak 180 juta orang. Namun, dengan jumlah vaksin yang masih terbatas, perlu dilakukan perhitungan dalam distribusi vaksin agar daerah dengan risiko COVID-19 yang tinggi dapat diprioritaskan sehingga risiko di daerah tersebut berkurang.

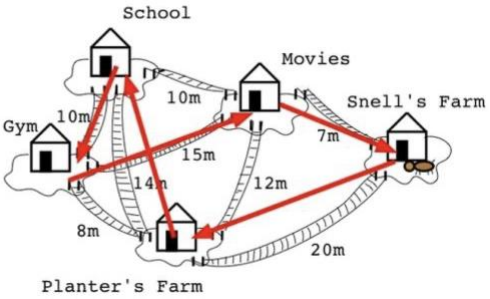
Persoalan distribusi vaksin ini dapat diilustrasikan sebagai *Travelling Salesman Problem* (TSP), yaitu permasalahan untuk mencari jalur dengan jarak terpendek pada suatu graf dan kembali ke titik asalnya. Provinsi dapat diibaratkan sebagai titik yang harus dikunjungi pada graf tersebut. Salah satu cara untuk menyelesaikan TSP adalah menggunakan algoritma *Branch and Bound* dengan matriks ongkos-tereduksi.

II. LANDASAN TEORI

A. *Travelling Salesman Problem (TSP)*

Travelling Salesman Problem (TSP) merupakan pesoalan yang berkaitan dengan kompleksitas komputasi yang diperoleh dari reduksi persoalan siklus Hamilton [4]. Definisi persoalan TSP adalah sebagai berikut. Pada suatu peta (graf) terdapat n buah kota (simpul) serta untuk setiap kota diketahui jarak (bobot) dengan kota-kota lainnya, kemudian akan dicari jarak terpendek (bobot terkecil) yang dapat ditempuh oleh seorang

pedagang sehingga pedagang tersebut melewati setiap kota tepat hanya sekali dan kembali lagi ke kota asalnya [5]. Persoalan TSP dapat diselesaikan menggunakan beberapa metode, diantaranya adalah *Brute Force*, *Branch and Bound*, dan Program dinamis.

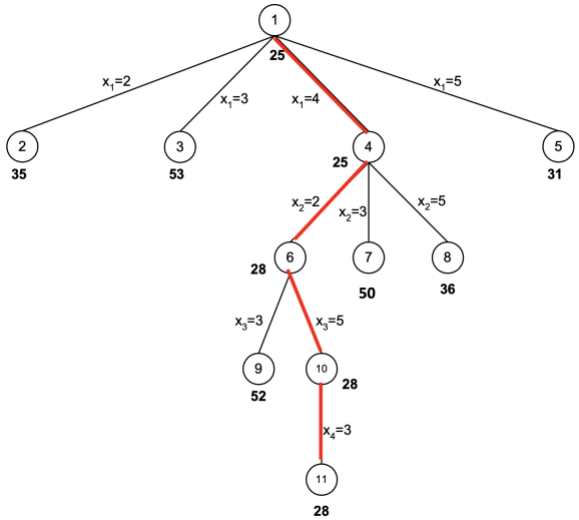


Gambar 2 Ilustrasi Travelling Salesman Problem (TSP)

(Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branchand-Bound-2021-Bagian2.pdf diakses pada tanggal 8 Mei 2021)

B. *Algoritma Branch and Bound*

Algortima *Branch and Bound* adalah salah satu algoritma yang digunakan untuk menyelesaikan suatu persoalan optimasi dengan cara meminimalkan atau memaksimalkan suatu fungsi objektif yang tidak melanggar batasan persoalan [6]. Algortima *Branch and Bound* diselesaikan menggunakan pohon ruang status yang pembangkitan simpul hidupnya mengikuti aturan FIFO pada sebuah *priority queue* dengan simpul yang memiliki nilai *cost* paling kecil / besar menjadi prioritas bergantung pada tujuan penggunaan algoritma tersebut.



Gambar 3 Ilustrasi Pohon Ruang Status Algoritma Branch and Bound

(Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf diakses pada tanggal 8 Mei 2021)

Cost suatu simpul pada pohon ruang status algoritma *Branch and Bound* pada umumnya merupakan taksiran yang mengikuti fungsi berikut.

$$c(i) = f(i) + g(i)$$

c(i) = ongkos untuk simpul i

f(i) = ongkos untuk mencapai simpul i dari akar

g(i) = ongkos untuk mencapai simpul tujuan dari simpul i

Langkah kerja algoritma *Branch and Bound* adalah sebagai berikut [6].

1. Masukkan simpul akar ke dalam sebuah *priority queue* Q. Jika simpul akar adalah simpul solusi, maka solusi telah ditemukan dan pencarian dihentikan.
2. Jika Q kosong, maka artinya tidak ada solusi dan pencarian dihentikan.
3. Jika Q tidak kosong, pilih dari antrian Q simpul *i* dengan *cost c(i)* yang paling kecil, jika terdapat beberapa simpul dengan *cost* terkecil yang sama, pilih sesuai ketentuan awal (bisa berdasarkan abjad atau ketentuan lainnya).
4. Jika simpul *i* adalah solusi, maka artinya solusi sudah ditemukan dan pencarian dihentikan. Jika *i* bukan solusi maka bangkitkan semua anak simpul *i* tersebut. Jika simpul *i* tidak memiliki anak, kembali ke langkah 2.
5. Untuk tiap anak *j* dari simpul *i*, hitung *cost c(j)*, lalu masukkan semua anak simpul *i* tersebut ke dalam *priority queue* Q.
6. Kembali ke langkah 2.

C. *Algoritma Branch and Bound pada TSP*

Penyelesaian TSP menggunakan *Branch and Bound* dapat dilakukan dengan dua cara yaitu menggunakan matriks ongkos-tereduksi dan menggunakan bobot tur lengkap. Pada artikel ini akan dibahas penyelesaian TSP dengan *Branch and Bound* menggunakan matriks ongkos-tereduksi. Matriks ongkos-tereduksi adalah *adjacency matrix* (matriks ketetanggaan) yang berisi ongkos antar setiap simpul pada suatu graf, yang kemudian baris dan kolom pada matriks tersebut direduksi sehingga setiap baris dan kolom memiliki setidaknya satu buah angka nol dengan elemen-elemen lainnya bukan bilangan negatif [5].

Gambar 4 Contoh Matriks Tereduksi

(Sumber : https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branchand-Bound-2021-Bagian2.pdf diakses pada tanggal 8 Mei 2021)

Langkah kerja penyelesaian TSP menggunakan algoritma *Branch and Bound* adalah sebagai berikut.

1. Menentukan matriks ketetanggaan berisi ongkos antar simpul suatu graf.

2. Mengubah matriks tersebut menjadi matriks ongkos-tereduksi dengan total pengurang elemen baris dan kolom dari matriks menjadi *cost* untuk simpul akar pada pohon ruang status (simpul akar adalah simpul tempat dimulainya TSP).
3. Membangkitkan simpul-simpul anak dari simpul akar dan menghitung *cost* untuk setiap simpul anak tersebut. Simpul-simpul anak kemudian dimasukkan ke dalam sebuah *priority queue* dengan simpul yang *cost-nya* paling kecil diprioritaskan.
4. Membangkitkan simpul selanjutnya dengan cara melakukan pengambilan *Head* dari *priority queue* tersebut. Jika yang terambil bukan merupakan simpul tujuan, maka bangkitkan simpul-simpul anak dari simpul tersebut, hitung *cost*-nya lalu masukkan ke *priority queue* (ulangi langkah ini hingga mencapai daun).
5. Jika sudah mencapai daun, bandingkan *cost* saat ini dengan *cost* simpul-simpul lainnya. Jika ada simpul dengan *cost* lebih besar maka simpul tersebut dibunuh karena simpul tersebut tidak akan mungkin menghasilkan *cost* yang lebih kecil dari *cost* saat ini.
6. Jika masih ada simpul yang belum dibunuh (*cost* masih lebih kecil dibandingkan *cost* saat ini), maka bangkitkan simpul tersebut hingga mencapai daun, lalu ulangi langkah 5.
7. Jika semua simpul sudah tidak ada simpul yang dapat dibangkitkan lagi, maka solusi TSP sudah ditemukan.

Cara perhitungan *cost* simpul anak pada pohon ruang status algoritma *Branch and Bound* dalam menyelesaikan TSP adalah sebagai berikut. (Misalkan pada sebuah matriks A akan dicari *cost* simpul *j* yang merupakan anak dari simpul *i*)

1. Ubah semua nilai pada baris *i*, dan kolom *j* menjadi ∞.
2. Ubah *A(j, i)* menjadi ∞ agar sisi *(j, i)* tidak digunakan sehingga tidak menyebabkan *looping*.
3. Reduksi semua baris dan kolom pada matriks A tersebut kecuali untuk elemen yang nilainya ∞.
4. Hitung *cost* simpul anak dengan menggunakan persamaan

$$c(S) = c(R) + A(i, j) + r$$

c(S) = bobot perjalanan minimum yang melalui simpul S

c(R) = bobot perjalanan minimum yang melalui simpul R, yang dalam hal ini simpul R adalah orangtua dari simpul S

A(i, j) = bobot sisi *(i, j)*

r = jumlah semua pengurang baris dan kolom matriks pada saat proses reduksi matriks untuk simpul S

III. PEMBAHASAN

A. Hubungan TSP dengan Distribusi Vaksin COVID-19

Distribusi vaksin COVID-19 dapat dianggap sebagai persoalan TSP dengan cara mengibaratkan distributor vaksin sebagai *salesman* yang harus mengunjungi *n* provinsi untuk mendistribusikan vaksin dan kembali lagi ke provinsi asalnya. Provinsi asal dalam hal ini adalah provinsi tempat vaksin disimpan. Jumlah vaksin yang masih terbatas membuat distribusi vaksin tersebut harus dilakukan dengan optimal agar provinsi yang memiliki risiko penyebaran COVID-19 lebih tinggi bisa memperoleh vaksin lebih dahulu sebelum provinsi dengan risiko penyebaran yang lebih rendah. Namun prioritas pengiriman vaksin tersebut tetap memperhatikan jarak antar provinsi agar biaya yang digunakan untuk distribusi vaksin tersebut tetap efektif.

Untuk menyelesaikan persoalan TSP menggunakan *Branch and Bound*, harus dibuat sebuah matriks ketetanggaan yang menyatakan bobot antar dua simpul. Pada persoalan distribusi vaksin ini, penulis membuat formula sebagai berikut untuk menentuan bobot antar dua simpul. Misalkan akan dicari bobot dari provinsi *i* ke provinsi *j*, maka

$$N_{ij} = \frac{1}{\frac{jumlah\ kasus\ positif\ COVID-19\ provinsi\ j}{jumlah\ penduduk\ provinsi\ j}} \ x \ \frac{jarak\ ij}{1000}$$

Formula tersebut diperoleh melalui pertimbangan sebagai berikut.

1. Pembagian jumlah COVID-19 di suatu provinsi dengan jumlah penduduk provinsi tersebut dilakukan untuk memperoleh perbandingan yang proporsional pada setiap provinsi antara kasus positif COVID-19 dengan jumlah penduduknya. Hasilnya kemudian dibuat berbanding terbalik agar ketika semakin besar persentase orang yang positif COVID-19, bobot menjadi semakin kecil. Artinya, semakin kecil bobot prioritas distribusi vaksin semakin tinggi.
2. Perkalian dengan jarak berfungsi untuk mempertimbangkan ongkos jarak antara suatu provinsi dengan provinsi selanjutnya, karena provinsi yang lebih jauh tentunya memiliki biaya perjalanan yang lebih besar. Jarak antar dua provinsi adalah jarak antara kedua ibukota provinsi tersebut.
3. Pembagian dengan 1000 dilakukan untuk mempermudah visualisasi agar bilangannya tidak terlalu besar.

Contoh dari penggunaan formula tersebut adalah sebagai berikut.

1. Distributor vaksin sedang berada di provinsi a, dan provinsi selanjutnya adalah provinsi b dan c. Provinsi a dan b berjarak 2000 km. Provinsi a dan c juga berjarak 2000 km.
2. Jumlah kasus COVID-19 di provinsi b sebanyak 5000 kasus dengan jumlah penduduk sebanyak 200000 orang. Jumlah kasus COVID-19 di provinsi c

juga sebanyak 5000 kasus tetapi dengan jumlah penduduk sebanyak 300000 orang

3. Berdasarkan formula tersebut, bobot dari provinsi a ke b pada matriks adalah sebesar

$$N_{ab} = \frac{1}{\frac{5000}{200000}} \; x \; \frac{2000}{1000} = 80$$

Sedangkan bobot dari provinsi a ke provinsi c adalah sebesar

$$N_{ac} = \frac{1}{\frac{5000}{300000}} \; x \; \frac{2000}{1000} = 120$$

Bobot provinsi a ke provinsi b lebih kecil dibandingkan dengan bobot provinsi a ke provinsi c karena dengan jarak dan jumlah kasus yang sama, persentase orang yang terpapar COVID-19 di provinsi b lebih besar dibandingkan dengan di provinsi c sehingga provinsi b mendapatkan prioritas yang lebih tinggi dibandingkan provinsi c.

B. Pengolahan Data

Pada artikel ini penulis akan memilih 6 provinsi di Indonesia yang akan dijadikan simpul dalam persoalan TSP. Pemilihan provinsi-provinsi tersebut adalah dengan mengambil masing-masing satu provinsi dengan kasus COVID-19 terbanyak dari pulau Sumatera, Jawa, Kalimantan, Sulawesi, Papua, dan kepulauan Sunda Kecil. Namun khusus pulau Jawa, diambil provinsi Jawa Barat sebagai simpul asal karena Jawa Barat merupakan tempat vaksin-vaksin COVID-19 disimpan.

Bedasarkan hal tersebut, provinsi terpilih yang diperoleh dari masing-masing pulau / kepulauan beserta jumlah penduduk [7] dan data kasus positif COVID-19 [8] adalah sebagai berikut (data kasus COVID-19 diambil pada tanggal 10 Mei 2021 pukul 14.40; dan data jumlah penduduk merupakan data proyeksi tahun 2020)

No	Provinsi	Jumlah Penduduk	Jumlah Kasus Positif COVID-19
1	Jawa Barat	49,565,200	291,030
2	Riau	6,951,200	48,432
3	Kalimantan Timur	3,664,700	69,493
4	Sulawesi Selatan	8,888,800	61,625
5	Bali	4,414,400	45,655
6	Papua	3,393,100	20,411

Tabel 1 Daftar Provinsi Uji

(Sumber : Dokumen pribadi)

Setelah mendapatkan provinsi yang akan digunakan untuk pengujian, langkah selanjutnya adalah menghitung jarak antara dua provinsi untuk setiap provinsi tersebut. Jarak antara dua provinsi diasumsikan adalah jarak antara ibukota kedua provinsi tersebut. Berikut adalah data jarak antara dua provinsi untuk masing-masing provinsi dalam kilometer [9]

	Jawa Barat	Riau	Kalimantan Timur	Sulawesi Selatan	Bali	Papua
Jawa Barat	-	1075.55	1277.99	1322.28	861.22	3703.06
Riau	1075.55	-	1750.04	2093.96	1837.33	4381.51
Kalimantan Timur	1277.99	1750.04	-	574.25	931.95	2631.83
Sulawesi Selatan	1322.28	2093.96	574.25	-	606.68	2382.1
Bali	861.22	1837.33	931.95	606.68	-	2902.66
Papua	3703.06	4381.51	2631.83	2382.1	2902.66	-

Tabel 2 Jarak Antar Provinsi (km)

(Sumber : Dokumen pribadi)

Menggunakan data tersebut dan formula untuk menghitung bobot dalam matriks ketetangaan, diperoleh matriks ketetanggaan sebagai berikut (hasil perhitungan dibulatkan ke bilangan bulat terdekat).

$$M = \begin{bmatrix} \infty & 154 & 67 & 191 & 83 & 615 \\ 183 & \infty & 92 & 302 & 178 & 728 \\ 218 & 251 & \infty & 89 & 90 & 438 \\ 225 & 300 & 30 & \infty & 58 & 396 \\ 146 & 264 & 49 & 88 & \infty & 483 \\ 630 & 629 & 138 & 344 & 281 & \infty \end{bmatrix}$$

Baris pada matriks tersebut adalah simpul asal sedangkan kolom adalah simpul tujuan. Urutan pengisian matriks sesuai dengan urutan pada tabel 1 yaitu Jawa Barat, Riau, Kalimantan Timur, Sulawesi Selatan, Bali, dan Papua. Contohnya bobot baris 1 kolom 2 pada matriks / M(1, 2) = 154 adalah bobot dari Provinsi Jawa Barat ke Provinsi Riau. Lokasi provinsi-provinsi yang dijadikan provinsi uji di peta Indonesia Indonesia terdapat pada gambar berikut.

Gambar 5 Lokasi 6 Provinsi Uji di Peta Indonesia

(Sumber : Dokumen pribadi)

C. Aplikasi Algoritma Branch and Bound dalam Program Sederhana

Setelah memperoleh matriks ketetanggaan, penulis melakukan uji coba penyelesaian TSP menggunakan algoritma *Branch and Bound* dengan cara membuat program sederhana menggunakan Bahasa Python 3. Program ini menerima masukkan berupa matriks ketetanggaan dengan ukuran tertentu yang sudah ditetapkan sebelumnya, dalam kasus ini, ukuran matriks adalah 6 x 6. Keluaran program ini adalah jalur optimal yang dapat dilalui dan total *cost* jalur tersebut. Langkah kerja program adalah sebagai berikut.

Program menerima masukkan berupa matriks ketetanggaan. Gambar berikut merupakan matriks ketetanggaan yang menjadi masukkan program.

X	154	67	191	83	615
183	X	92	302	178	728
218	251	X	89	90	438
225	300	30	X	58	396
146	264	49	88	X	483
630	629	138	344	281	X

Gambar 6 Matriks Masukan Program

(Sumber : Dokumen pribadi)

Pada matriks masukan tersebut, 'X' digunakan sebagai pengganti tanda ∞.

Setelah menerima matriks ketetanggaan, program melakukan reduksi terhadap matriks tersebut. Fungsi untuk melakukan reduksi matriks adalah sebagai berikut.

```
def reduksiMatriks(matriks):
    nrow = len(matriks)
    ncol = len(matriks[0])
    r = 0
    for i in range(nrow):
        if(not(rowContainsZero(matriks, i))):
            subtractor = minimumValueRow(matriks, i)
            matriks = reduksiBaris(matriks, i, subtractor)
            r += subtractor

    for j in range(ncol):
        if(not(colContainsZero(matriks, j))):
            subtractor = minimumValueCol(matriks, j)
            matriks = reduksiKolom(matriks, j, subtractor)
            r += subtractor

    return matriks, r
```

Gambar 7 Fungsi Reduksi Matriks

(Sumber : Dokumen pribadi)

Fungsi tersebut mengembalikan matriks ketetanggaan yang sudah direduksi menjadi matriks ongkos-tereduksi dan mengembalikan jumlah pengurang baris dan kolom (*r*) yang akan menjadi *cost* untuk simpul akar. Matriks ongkos-tereduksi yang diperoleh dan *cost* simpul akar adalah sebagai berikut.

X	0	0	124	15	199
0	X	0	210	85	287
38	75	X	0	0	0
104	183	0	X	27	17
6	128	0	39	X	85
401	404	0	206	142	X

Cost akar = 993

Gambar 8 Matriks Ongkos-tereduksi dan Cost Simpul Akar

(Sumber : Dokumen pribadi)

Langkah selanjutnya program akan mengekspan simpul-simpul anak hingga ditemukannya jalur optimal dari persoalan TSP tersebut. Fungsi yang digunakan untuk mengekspan simpul-simpul anak adalah sebagai berikut.

```
def ekspan(matriks, queue):
    curr_parent = getHead(queue)
    curr_matriks = curr_parent.matriksState

    listOfChild = getChild(curr_parent)
    for child in listOfChild:
        if(not(isVisited(child))):
            child.cost, child.matriksState = getCost(curr_parent, child, curr_matriks)
            queue = insertQueue(child)

    queue = sortPriority(queue)
    return curr_matriks, queue, curr_parent
```

Gambar 9 Fungsi Ekspan Simpul

(Sumber : Dokumen pribadi)

Fungsi tersebut akan mengembalikan kondisi matriks saat ini, queue, dan curr_parent yang merupakan simpul hidup saat ini, yang diambil dari *Head* queue. Sementara itu, struktur data queue dibuat menggunakan list yang di-*sort* sesuai dengan *cost*-nya. Simpul dengan *cost* paling kecil akan ditempatkan di indeks paling rendah dalam list tersebut.

Ekspan terhadap suatu simpul akan terus dilakukan hingga solusi ditemukan. Ketika solusi sudah ditemukan, program akan tetap melakukan pengecekan terhadap isi dari queue. Simpul di dalam queue yang *cost*-nya lebih besar dari *cost* saat ini akan dibunuh dengan cara dikeluarkan dari queue. Apabila ketika dilakukan pengecekan ditemukan simpul dengan *cost* yang lebih kecil dari *cost* sekarang, maka simpul dengan *cost* yang lebih kecil tersebut akan diekspan lagi. Pengecekan tersebut dilakukan menggunakan algoritma berikut.

```
def bunuhSimpul(queue):
    for simpul in queue:
        if(simpul.cost > curr_cost):
            queue.remove(simpul)
    return queue
```

Gambar 10 Fungsi Bunuh Simpul

(Sumber : Dokumen pribadi)

```
def doneStatus(queue, curr_parent, matriks):
    if(len(curr_parent.visited) == len(matriks)):
        curr_solution = curr_parent
        curr_cost = curr_parent.cost
        queue = bunuhSimpul(queue)
        if(len(queue) == 0):
            return True, queue
    return False, queue
```

Gambar 11 Fungsi Status Pencarian

(Sumber : Dokumen pribadi)

Ketika queue sudah habis, artinya solusi sudah ditemukan dan persoalan TSP sudah selesai. Jika setelah simpul-simpul dibunuh masih terdapat simpul yang *cost*-nya lebih kecil dari *cost* saat ini maka simpul tersebut akan diekspan lagi menggunakan fungsi yang terdapat pada gambar 9.

Hasil yang penulis dapatkan dari pencarian rute distribusi vaksin COVID-19 menggunakan TSP dengan algoritma *Branch and Bound* pada program sederhana tersebut adalah sebagai berikut.

```
Hasil Pencarian :

Jalur optimal :  1 5 4 6 3 2 1
Cost : 1139
```

Gambar 12 Hasil Pencarian Rute

(Sumber : Dokumen pribadi)

Rute optimal yang ditemukan menggunakan algoritma *Branch and Bound* yang dimulai dari provinsi 1 adalah 1 – 5 – 4 – 6 – 3 – 2 – 1 dengan total *cost* sebesar 1139. Berdasarkan urutan provinsi sebelumnya, maka rute sebenarnya adalah Jawa Barat – Bali – Sulawesi Selatan – Papua – Kalimantan Timur – Riau – Jawa Barat.

Gambar 12 Gambaran Rute Optimal dalam Peta

(Sumber : Dokumen pribadi)

IV. KESIMPULAN

Persoalan prioritas distribusi vaksin COVID-19 dapat dimodelkan menggunakan konsep permasalahan *Travelling Salesman Problem* (TSP) yang kemudian diselesaikan dengan Algoritma *Branch and Bound* menggunakan matriks ongkos-tereduksi. Pemanfaatan algoritma *Branch and Bound* untuk menyelesaikan TSP tidak terbatas pada persoalan distribusi vaksin COVID-19, tetapi juga dapat diterapkan pada persoalan-persoalan lainnya yang dapat dimodelkan dengan TSP.

Pemodelan masalah menggunakan TSP dan penyelesaian TSP menggunakan *Branch and Bound* merupakan cara yang efektif untuk graf dengan jumlah simpul yang terbatas. Apabila graf memiliki jumlah simpul yang banyak maka penyelesaian masalah dengan cara tersebut akan membutuhkan waktu yang lama.

TAUTAN VIDEO DI YOUTUBE

https://youtu.be/T8veFjda0e4

PENUTUP

Puji syukur penulis panjatkan kepada Allah karena berkat kehendak-Nya makalah ini dapat diselesaikan. Penulis menghaturkan terima kasih kepada seluruh pihak yang telah membantu penulis dalam penulisan makalah ini, diantaranya adalah dosen mata kuliah IF2211 Strategi Algoritma, khususnya kepada Bapak Prof. Dwi Hendratmo Widyantoro sebagai dosen penulis di kelas K-3, keluarga penulis, dan teman-teman penulis. Terakhir penulis mengucapkan terima kasih kepada pihak yang telah membagikan hasil tulisannya untuk penulis gunakan sebagai referensi dalam makalah ini. Penulis berharap makalah ini dapat bermanfaat untuk setiap orang yang ingin menggunakan atau mengaplikasikan algoritma *Branch and Bound* dalam berbagai permasalahan. Selain itu penulis juga berharap pandemi COVID-19 yang sedang melanda dunia cepat usai agar masyarakat dapat menjalani kehidupannya secara normal seperti dahulu.

REFERENSI

[1] https://www.who.int/emergencies/diseases/novel-coronavirus-2019/interactive-timeline (diakses pada tanggal 7 Mei 2021).

[2] https://www.who.int/health-topics/coronavirus#tab=tab_1 (diakses pada tanggal 7 Mei 2021).

[3] http://p2p.kemkes.go.id/program-vaksinasi-covid-19-mulai-dilakukan-presiden-orang-pertama-penerima-suntikan-vaksin-covid-19/ (diakses pada tanggal 8 Mei 2021).

[4] https://opendsa-server.cs.vt.edu/ODSA/Books/Everything/html/hamiltonianCycle_to_TSP.html (diakses pada tanggal 8 Mei 2021).

[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branchand-Bound-2021-Bagian2.pdf (diakses pada tanggal 8 Mei 2021).

[6] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf (diakses pada tanggal 8 Mei 2021).

[7] https://www.bps.go.id/indicator/12/1886/1/jumlah-penduduk-hasil-proyeksi-menurut-provinsi-dan-jenis-kelamin.html (diakses pada tanggal 10 Mei 2021).

[8] https://covid19.go.id/peta-sebaran-covid19 (diakses pada tanggal 10 Mei 2021).

[9] https://www.gps-coordinates.net/distance (diakses pada tanggal 10 Mei 2021).

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, 10 Mei 2021

Irvin Andryan Pratomo - 13519162