Penerapan Algoritma Branch and Bound pada Penentuan Prioritas Pengetesan Kontak Erat dengan Terkonfirmasi Positif COVID-19

Epata Tuah/13519120
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: epatatuah180901@gmail.com

***Abstrak*—Algoritma *Branch & Bound* merupakan salah satu algoritma yang banyak diterapkan pada permasalahan optimasi suatu permasalahan. Salah satu penerapannya adalah pada penanganan pandemi. COVID-19 merupakan penyakit yang disebabkan oleh virus SARS-CoV-2 dan menjadi penyebab pandemi dari tahun 2020 hingga waktu yang belum dipastikan, dengan tingkat penyebaran yang cukup tinggi, sehingga dibutuhkan mekanisme pengetesan COVID-19 yang tepat. Kondisi yang ideal adalah melacak dan mengetes semua orang yang telah berstatus kontak erat, namun kondisi di Indonesia saat ini memiliki ketimpangan infrastruktur kesehatan, sehingga diperlukan penentuan prioritas pengetesan kontak erat dengan terkonfirmasi positif COVID-19.**

Kata Kunci—COVID-19;* Kontak Erat*; Branch & Bound

I. Pendahuluan

COVID-19 merupakan penyakit sistem pernapasan menular yang disebabkan oleh virus SARS-CoV-2. Penyakit ini pertama kali dideteksi di Kota Wuhan, Tiongkok, pada bulan Desember 2019. Penyakit ini menimbulkan gejala utama pada sistem pernafasan seperti gejala flu, beberapa di antaranya yakni demam, batuk kering, nyeri tenggorokan, dan sesak nafas.[1] Penyakit ini memiliki tingkat kematian sekitar 2,74% di Indonesia[2]. Angka ini secara sekilas memang tampak kecil, namun COVID-19 sangat mudah menyebar, sehingga jika terjadi pembiaran dapat mengakibatkan penuhnya kapasitas tempat tidur rumah sakit secara cepat hingga *collapse*[3]. Jika fasilitas kesehatan *collapse*, akan banyak pembiaran penanganan penyakit yang sedang diderita masyarakat (tidak hanya berefek pada penderita COVID-19, namun juga penderita penyakit lainnya) sehingga tingkat kematian seluruh penyakit akan meningkat. Oleh karena itu, pengetesan kontak erat COVID-19 yang efektif dan efisien sangat diperlukan.

Kontak erat merupakan istilah baru yang dikenalkan Kementerian Kesehatan Republik Indonesia untuk menggantikan istilah ODP atau Orang Dalam Pemantauan. Menurut Kemenkes, Kontak Erat adalah orang yang mempunyai riwayat kontak dengan seseorang yang terkonfirmasi positif COVID-19[4]. Adapun rinciannya adalah kontak tatap muka/berdekatan, sentuhan fisik, orang yang memberikan perawatan langsung, serta situasi lainnya. Pengetesan kontak erat dilakukan dengan pelacakan terlebih dahulu, yakni mencatat waktu dan tempat, melakukan kunjungan terhadap seseorang dengan tersebut, dan melakukan tes usap/antigen.

Pada kondisi ideal, petugas lacak atau *contact tracer* harus melacak dan mengetes semua orang yang telah berstatus kontak erat dengan seseorang yang terkonfirmasi positif COVID-19[5]. Namun, Indonesia merupakan negara kepulauan dengan kondisi geografi yang tidak rata dan infrastruktur yang belum optimal, sehingga ada beberapa daerah yang memiliki keterbatasan *contact tracer* maupun alat tes usap/antigen[6]. Oleh karena itu, dibutuhkan algoritma untuk menentukan prioritas pengetesan kontak erat dengan seseorang yang terkonfirmasi positif COVID-19. Algoritma yang dipilih adalah algoritma *branch & bound* karena merupakan algoritma yang digunakan untuk persoalan optimisasi.

II. Landasan Teori

Algoritma *Branch and Bound* merupakan algoritma gabungan algoritma BFS dan penggunaan konsep *least cost search* (kasus minimasi) atau *highest cost search* (kasus maksimasi) [7].

Pada algoritma ini, setiap simpul diberikan suatu nilai *cost* $\hat{c}(i)$, yakni nilai taksiran lintasan termurah atau termahal ke simpul status tujuan yang melalui simpul status i. Simpul setelahnya yang akan diekspansi tidak lagi berdasarkan urutan pembangkitannya, tetapi simpul yang memiliki *cost* paling kecil (kasus minimasi) atau paling besar (kasus maksimasi) [7].

Algoritma *Branch and Bound* memberlakukan peringkasan jalur yang dianggap tidak lagi mengarah pada solusi. Kriteria peringkasan adalah

1. Nilai simpul tidak lebih baik dari nilai terbaik sejauh ini
2. Simpul tidak merepresentasikan solusi yang *feasible* karena ada batasan yang dilanggar
3. Solusi pada simpul tersebut hanya terdiri atas satu titik alias tidak ada pilihan lain, maka dibandingkan nilai fungsi objektif dengan solusi terbaik saat ini, dengan yang terbaik yang diambil[7].

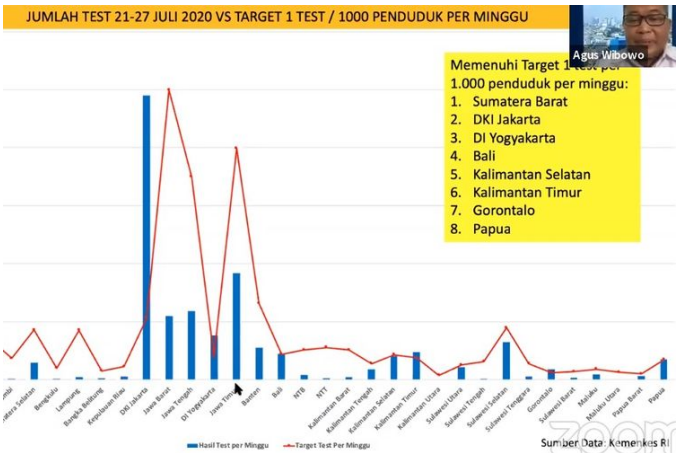
Pada umumnya, letak simpul solusi tidak diketahui, oleh karena itu, nilai *cost* simpul *i* merupakan estimasi ongkos

termurah lintasan dari simpul *i* ke simpul solusi (yang tidak diketahui letaknya), dilambangkan dengan ĉ(i), ĉ(i) dihitung secara heuristik. Dengan kata lain, ĉ(i) melambangkan batas bawah (*lower bound*) dari ongkos pencarian solusi dari status *i*.

Algoritma Global *Branch and Bound* adalah sebagai berikut:

1. Masukkan simpul akar ke dalam antrian Q. Jika simpul akar adalah simpul solusi (*goal node*), maka solusi telah ditemukan. Stop.
2. Jika Q kosong, tidak ada solusi. Stop.
3. Jika Q tidak kosong, pilih dari antrian Q simpul *i* yang mempunyai nilai *cost* ĉ(i) paling kecil. Jika terdapat beberapa simpul *i* yang memenuhi, pilih satu secara sembarang.
4. Jika simpul *i* adalah simpul solusi, berarti solusi sudah ditemukan, stop. Jika simpul *i* bukan simpul solusi, maka bangkitkan semua anak-anaknya. Jika *i* tidak mempunyai anak, Kembali ke Langkah 2.
5. Untuk setiap anak *j* dari simpul *i* , hitung ĉ(i) dan masukkan semua anak-anak tersebut ke dalam Q.
6. Kembali ke langkah 2[7].

III. PEMBAHASAN



Gambar 3.1 Grafik jumlah *testing* yang dilakukan periode 21-27 Juli 2020

(Sumber: Kemenkes RI)

Pada kondisi pandemi, pelacakan dan pengetesan adalah hal yang sangat penting dilakukan untuk mencegah penyebaran penyakit yang lebih luas. Namun, kondisi infrastruktur kesehatan setiap daerah berbeda-beda, sehingga seringkali terjadi gap antara jumlah tes yang dilakukan di pusat dan daerah. Sebagai contoh, jika dilihat pada gambar di bawah, pada rentang 21-27 Juli 2020 jumlah tes PCR yang dilakukan di Provinsi DKI Jakarta sangat timpang jika dibandingkan jumlah tes PCR yang dilakukan di provinsi lain. Hal ini tentu perlu diperhatikan dikarenakan adanya ketimpangan jumlah tes menyebabkan adanya ketidakakuratan data terkonfirmasi positif COVID-19 yang di-*publish* pemerintah. Oleh karena itu, pelacakan dan pengetesan kontak erat dengan yang terkonfirmasi positif COVID-19 sangat perlu untuk dilakukan.

Strategi *contact tracing* yang dilakukan adalah Identifikasi, Informasikan, dan Isolasi. Identifikasi adalah pencatatan waktu dan tempat dari orang-orang yang berkontak dengan penderita COVID-19. Informasikan adalah memberitahukan orang-orang yang memiliki kemungkinan untuk terpapar virus. Terakhir, Isolasi, yakni tidak keluar rumah untuk mencegah penyebaran yang lebih lanjut. Hal yang perlu digarisbawahi adalah pencatatan waktu dan tempat dari orang-orang yang berkontak dengan penderita COVID-19.

Strategi yang digunakan pada penggunaan algoritma *branch and bound* untuk mendapatkan prioritas orang yang akan dites adalah dengan mencatat waktu si kontak erat sebagai *profit* dan *weight*. Waktu yang perlu dicatat sebagai *profit* adalah waktu si kontak erat melakukan aktivitas di luar rumah setelah melakukan kontak dengan seseorang yang terkonfirmasi positif COVID-19 dan waktu yang perlu dicatat sebagai *weight* adalah waktu yang dibutuhkan petugas lacak untuk mengunjungi kediaman si kontak erat dan melakukan pengetesan.

Waktu seseorang melakukan aktivitas di luar rumah penting untuk dicatat dikarenakan semakin banyak aktivitas luar ruangan yang dilakukan, semakin rentan seseorang tersebut untuk menjadi *superspreader* virus COVID-19, sehingga diprioritaskan untuk dilacak dan dites terlebih dahulu, lalu melakukan pelacakan lagi ke kontak eratnya jika seseorang tersebut positif COVID-19. Selain itu, waktu yang dibutuhkan petugas lacak untuk mengunjungi kediaman si kontak erat juga penting untuk dicatat dikarenakan semakin dekat kediaman si kontak erat dengan petugas lacak, maka semakin efektif juga untuk dapat melakukan pengetesan kepada kontak erat sebanyak-banyaknya. Dua komponen ini perlu di maksimalkan sehingga mendapatkan prioritas pengetesan yang maksimal dan sebanyak-banyaknya.

Berikut merupakan contoh kasus uji penerapan algoritma *branch and bound* pada penentuan prioritas pengetesan kontak erat. Asumsi satu petugas lacak memiliki jam kerja 8 jam. Jam kerja tersebut dianggap sebagai kapasitas waktu maksimal yang dibutuhkan petugas lacak untuk berkunjung ke kontak erat.

Misal seorang Z terkonfirmasi positif COVID-19 setelah dites usap. Z dihubungi petugas lacak untuk mendaftarkan orang-orang yang telah melakukan kontak erat dengan si Z. Daftar orang-orang tersebut adalah sebagai berikut beserta waktu yang dibutuhkan untuk berkunjung ke kontak erat:

Nama	**Waktu total kontak erat melakukan aktivitas di luar rumah setelah melakukan kontak dengan Z (dilambangkan juga sebagai *profit* atau p)**	**Waktu yang dibutuhkan untuk berkunjung ke kontak erat (dilambangkan juga sebagai *weight* atau w)**	**p/w**
A	20 jam	7 jam	2,857
B	15 jam	2 jam	7,500
C	5 jam	1 jam	5,000
D	15 jam	4 jam	3,750

E	8 jam	3 jam	2,667
F	13 jam	3 jam	4,333

Tabel 3.1 Data Kontak Erat

p/w setiap objek diurutkan menurut agar pencarian solusi dapat lebih mangkus.

i	**Pi/wi**
1	7,500
2	5,000
3	4,333
4	3,750
5	2,857
6	2,667

Tabel 3.2 Urutan rasio p/w

Waktu yang perlu dimaksimasi adalah waktu si kontak erat melakukan aktivitas di luar. Rumus *upper bound* atau *cost* menggunakan konsep seperti pada permasalahan *branch and bound* 1/0 Knapsack Problem. Simpul pohon ruang status berikutnya yang diekspansi adalah simpul hidup yang memiliki *cost* yang paling besar. Berikut rumus *upper bound* yang digunakan pada algoritma ini.

$$\hat{c}(i) = F + (K - W)p_{i+1}/w_{i+1}$$

Pohon ruang status berbentuk pohon biner dengan cabang kiri menyatakan objek *i* yang dipilih ($x_i = 1$) dan cabang kanan menyatakan objek *i* yang tidak dipilih ($x_i = 0$). Setiap simpul diisi dengan total *weight* waktu yang sudah diambil (W) dan total waktu aktivitas yang sudah dicapai (F).

Langkah pertama adalah membangkitkan simpul akar (simpul 0) terlebih dahulu. Selanjutnya, ekspansi simpul hingga menemui solusi. Berikut Langkah-langkah per simpul.

Simpul 0: W = 0, F = 0

$$\hat{c}(0) = F + (K - W)(p_1/w_1) = 0 + (8 - 0)(7{,}500) = 60$$

Simpul 1: W = 0 + 7 = 7, F = 0 + 20 = 20 (x1 diambil)

$$\hat{c}(1) = F + (K - W)(p_2/w_2) = 20 + (8 - 7)(5{,}000) = 25$$

Simpul 2: W = 0 + 0 = 0, F = 0 + 0 = 0 (x1 tidak diambil)

$$\hat{c}(2) = F + (K - W)(p_2/w_2) = 0 + (8 - 0)(5{,}000) = 40$$

Simpul 2 memiliki *cost* paling besar, sehingga simpul 2 yang diekspansi selanjutnya

Simpul 3: W = 0 + 2 = 2, F = 0 + 15 = 15 (x2 diambil)

$$\hat{c}(3) = F + (K - W)(p_3/w_3) = 15 + (8 - 2)(4{,}333) = 41$$

Simpul 4: W = 0 + 0 = 0, F = 0 + 0 = 0 (x2 tidak diambil)

$$\hat{c}(4) = F + (K - W)(p_3/w_3) = 0 + (8 - 0)(4{,}333) = 34{,}667$$

Simpul 3 memiliki *cost* paling besar, sehingga simpul 3 yang diekspansi selanjutnya

Simpul 5: W = 2 + 1 = 3, F = 15 + 5 = 20 (x3 diambil)

$$\hat{c}(5) = F + (K - W)(p_4/w_4) = 20 + (8 - 3)(3{,}750) = 38{,}750$$

Simpul 6: W = 2 + 0 = 2, F = 15 + 0 = 15 (x3 tidak diambil)

$$\hat{c}(6) = F + (K - W)(p_4/w_4) = 15 + (8 - 2)(3{,}750) = 37{,}500$$

Simpul 5 memiliki *cost* paling besar, sehingga simpul 5 yang diekspansi selanjutnya

Simpul 7: W = 3 + 4 = 7, F = 20 + 15 = 35 (x4 diambil)

$$\hat{c}(7) = F + (K - W)(p_5/w_5) = 35 + (8 - 7)(2{,}857) = 37{,}857$$

Simpul 8: W = 3 + 0 = 3, F = 20 + 0 = 20 (x4 tidak diambil)

$$\hat{c}(8) = F + (K - W)(p_5/w_5) = 20 + (8 - 3)(2{,}857) = 34{,}285$$

Simpul 7 memiliki *cost* paling besar, sehingga simpul 7 yang diekspansi selanjutnya

Simpul 9: W = 7 + 3 = 10 > kapasitas waktu maksimal (8) (x5 diambil)

Simpul 9 dibunuh karena jumlah W melebihi kapasitas waktu petugas lacak maksimum.

Simpul 10: W = 7 + 0 = 7, F = 35 + 0 = 35 (x5 tidak diambil)

$$\hat{c}(10) = F + (K - W)(p_6/w_6) = 35 + (8 - 7)(2{,}667) = 37{,}667$$

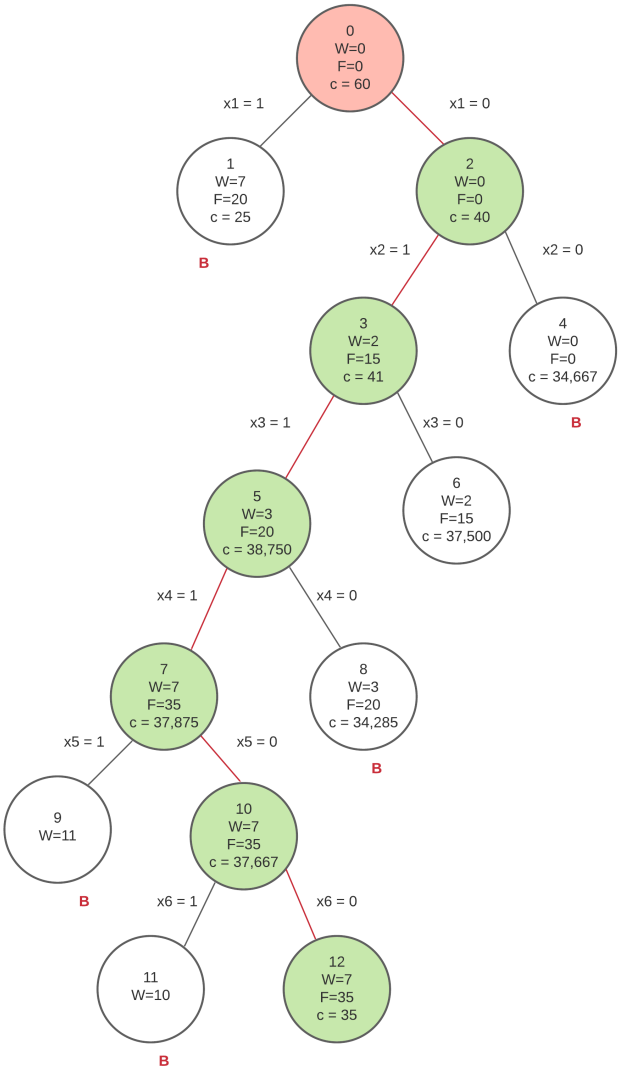
Simpul 10 memiliki *cost* paling besar, sehingga simpul 10 yang diekspansi selanjutnya.

Simpul 11 (w6 diambil) : W = 7 + 3 = 10 > 8

Simpul 11 dibunuh karena jumlah W melebihi kapasitas waktu petugas lacak maksimum.

Simpul 12: W = 7 + 0 = 7, F = 35 + 0 = 35 (x6 tidak diambil)

$$\hat{c}(12) = F + (K - W)(p_7/w_7) = 35 + (8 - 7)(0) = 35$$



Gambar 3.2 *best solution so far* pertama

Simpul 12 adalah simpul solusi (*best solution so far*), semua simpul hidup yang *cost*-nya lebih kecil dari 35 dibunuh. Dari pohon ruang status, tersisa simpul 6 sehingga simpul 6 diekspansi.

Simpul 13: W = 2 + 4 = 6, F = 15 + 15 = 30 (x4 diambil)

$$\hat{c}(13) = F + (K - W)(p_5/w_5) = 30 + (8 - 6)(2{,}857) = 35{,}714$$

Simpul 14: W = 2 + 0 = 2, F = 15 + 0 = 15 (x4 tidak diambil)

$$\hat{c}(14) = F + (K - W)(p_5/w_5) = 15 + (8 - 2)(2{,}857) = 32{,}142$$

Simpul 13 memiliki *cost* paling besar, sehingga simpul 13 yang diekspansi selanjutnya.

Simpul 15 (x5 diambil) : W = 6 + 3 = 9 > 8

Simpul 15 dibunuh karena jumlah W melebihi kapasitas waktu petugas lacak maksimum.

Simpul 16: W = 6 + 0 = 6, F = 30 + 0 = 30 (x5 tidak diambil)

$$\hat{c}(16) = F + (K - W)(p_6/w_6) = 30 + (8 - 6)(2{,}667) = 35{,}334$$

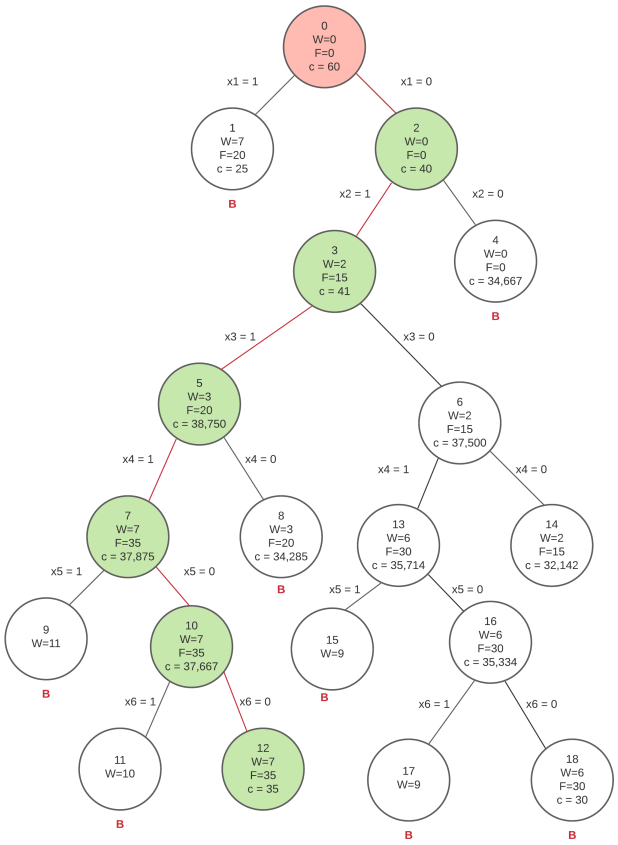
Simpul 16 memiliki *cost* paling besar, sehingga simpul 16 yang diekspansi selanjutnya.

Simpul 17 (w6 diambil) : W = 6 + 3 = 9 > 8

Simpul 17 dibunuh karena jumlah W melebihi kapasitas waktu petugas lacak maksimum.

Simpul 18: W = 6 + 0 = 6, F = 30 + 0 = 30 (x6 tidak diambil)

$$\hat{c}(18) = F + (K - W)(p_7/w_7) = 30 + (8 - 6)(0) = 30$$



Gambar 3.3 *best solution so far* kedua

Simpul 18 adalah simpul solusi, namun karena nilai *cost* lebih kecil dari nilai *cost* solusi sebelumnya, simpul solusi dibunuh. Semua simpul hidup yang *cost*-nya lebih kecil dari 30 dibunuh.

Dari pohon ruang status, tersisa simpul 14 sehingga simpul 14 diekspansi

Simpul 19: W = 2 + 3 = 5, F = 15 + 8 = 23 (x5 diambil)

$$\hat{c}(19) = F + (K - W)(p_6/w_6) = 23 + (8 - 5)(2{,}667) = 31$$

Simpul 20: W = 2 + 0 = 2, F = 15 + 0 = 15 (x5 tidak diambil)

$$\hat{c}(20) = F + (K - W)(p_6/w_6) = 15 + (8 - 2)(2{,}667) = 31$$

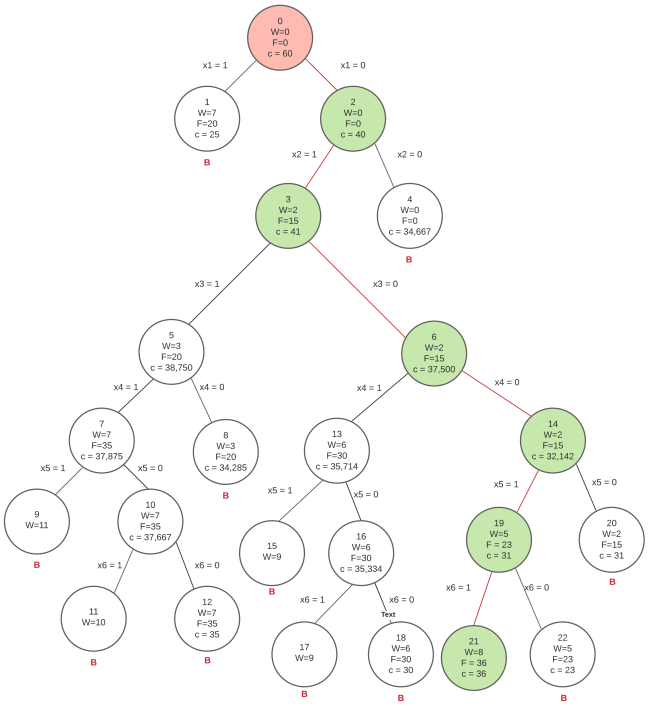
Simpul 19 dan 20 memiliki *cost* yang sama besar, maka dipilih sembarang. Simpul 19 yang menjadi sisi kiri dipilih.

Simpul 21: W = 5 + 3 = 8, F = 23 + 13 = 36 (x6 diambil)

$$\hat{c}(21) = F + (K - W)(p_6/w_6) = 36 + (8 - 8)(0) = 36$$

Simpul 22: W = 5 + 0 = 5, F = 23 + 0 = 23 (x6 tidak diambil)

$$\hat{c}(22) = F + (K - W)(p_6/w_6) = 23 + (8 - 5)(0) = 23$$



Gambar 3.4 *best solution so far* ketiga

Simpul 21 adalah simpul solusi, semua simpul hidup yang *cost*-nya lebih kecil dari 36 dibunuh. Simpul hidup habis, sehingga solusinya adalah (0, 1, 0, 0, 1, 1) atau B->E->F. Dari solusi tersebut, petugas lacak akan memprioritaskan B terlebih dahulu, lalu E, terakhir F. Solusi yang pernah terbentuk adalah sebagai berikut.

i	Solusi	*Cost* akhir	Status
0	B->C->D	35	Tidak Optimal
1	B->D	30	Tidak Optimal
2	B->E->F	36	Optimal

Tabel 3.3 Daftar solusi yang pernah terbentu

IV. IMPLEMENTASI

Implementasi yang saya buat menggunakan bahasa pemrograman Java. Ada 3 *source code* yang dibuat pada implementasi, yakni TracingBranchBound.java, Node.java, dan KontakErat.java. *Source code* yang ditampilkan hanyalah sebagian *source code* TracingBranchBound.java untuk memberikan gambaran algoritma *branch and bound* yang dipakai. Adapun *source code* dari algoritma *branch and bound* pada TracingBranchBound.java adalah sebagai berikut.

```
//Algoritma Branch and Bound untuk Prioritas Tracing Pengetesan Kontak Erat
public TracingBranchBound(KontakErat[] kontakErats, int kapasitasWaktu){
    Node temp1, temp2;
    int i = 0;
    int level = 0;
    int solutionCount = 0;
    boolean multipleSolution = false;
    List<Node> multipleSolutionNodes = new ArrayList<>();
    List<Integer> multipleSolutionIdx = new ArrayList<>();

    //Menambahkan rasio
    for (KontakErat kE : kontakErats){
        rasios.add((double) kE.getTotalWaktuAktivitas()/kE.getTotalWaktuBerkunjung());
    }

    //rasio terakhir berupa angka nol
    rasios.add(0.0);

    //Mengurutkan rasio menurun
    Collections.sort(rasios, Collections.reverseOrder());

    //Inisialisasi simpul akar (W=0, F=0)
    double costZero = upperBound(0, kapasitasWaktu, 0, rasios.get(i));
    this.currNode = new Node("initial",0,0, costZero, level, -1);
    activeNodes.add(currNode);

    while (!activeNodes.isEmpty()){ //Selama masih ada simpul aktif
        i+=1;
        level+=1;
        //Pengecekan cost cabang pohon sisi kiri
        double currWeight1 = this.currNode.getCurrWeight() +
                Arrays.stream(kontakErats).
                        toList().
                        get(this.currNode.getLevel()).
                        getTotalWaktuBerkunjung();
        double currProfit1 = this.currNode.getCurrProfit() +
                Arrays.stream(kontakErats).
                        toList().
                        get(this.currNode.getLevel()).
                        getTotalWaktuAktivitas();
        double currCost1 = upperBound(currProfit1, kapasitasWaktu, currWeight1, rasios.get(i));
        temp1 = new Node(Arrays.stream(kontakErats).
                toList().
                get(this.currNode.getLevel()).
                getNama(), currProfit1, currWeight1, currCost1, level, 1);

        //tambahkan ke array activeNodes terlebih dahulu
        activeNodes.add(temp1);

        //bunuh jika currWeight1 melebihi kapasitasWaktu
        if (currWeight1 > kapasitasWaktu) {
            currCost1 = -1;
            activeNodes.remove(temp1);
        }
```

Gambar 4.1 Algoritma *branch and bound* bagian 1

```
//Bandingkan nilai Cost
if (currCost1 > currCost2){ //cost sisi kiri lebih besar
    activeNodes.remove(currNode);
    activeNodes.remove(temp1);
    currNode = temp1;
    activeNodes.add(currNode);
    if (solutionCount > 0){
        tempSolutionNodes.add(currNode);
        searchSolutionNodes.add(currNode);
    } else {
        bestSolutionNodes.add(currNode);
        multipleSolutionNodes.add(currNode);
    }
} else if (currCost1 < currCost2) { //cost sisi kanan lebih besar
    activeNodes.remove(currNode);
    activeNodes.remove(temp2);
    currNode = temp2;
    activeNodes.add(currNode);
    if (solutionCount > 0){
        tempSolutionNodes.add(currNode);
        searchSolutionNodes.add(currNode);
    } else {
        bestSolutionNodes.add(currNode);
        multipleSolutionNodes.add(currNode);
    }
} else { //cost kedua sisi sama
    //ambil sembarang
    //dipilih untuk mengambil node kiri terlebih dahulu (bernilai 1/true alias pilihan diambi
    activeNodes.remove(currNode);
    activeNodes.remove(temp1);
    currNode = temp1;
    activeNodes.add(currNode);
    if (solutionCount > 0){
        tempSolutionNodes.add(currNode);
        searchSolutionNodes.add(currNode);
    } else {
        bestSolutionNodes.add(currNode);
        multipleSolutionNodes.add(currNode);
    }
}
```

Gambar 4.2 Algoritma *branch and bound* bagian 2

```
//jika level node sudah paling bawah
if (level == kontakErats.length){
    if (solutionCount > 0){ //kode untuk mengeluarkan output sesuai urutan cost paling maksim
        if (tempSolutionNodes.
                get(tempSolutionNodes.size()-1).
                getCurrCost() > bestSolutionNodes.
                get(bestSolutionNodes.size()-1).
                getCurrCost()){
            multipleSolution = false;
            List<Node> solNodesPartition = new ArrayList<>();
            for (Node node : searchSolutionNodes){
                if (node.getLevel() < tempSolutionNodes.get(0).getLevel()){
                    solNodesPartition.add(node);
                }
            }
            for (Node node : tempSolutionNodes){
                solNodesPartition.add(node);
            }
            bestSolutionNodes.clear();
            for (Node node : solNodesPartition){
                bestSolutionNodes.add(node);
                multipleSolutionNodes.add(node);
            }
            tempSolutionNodes.clear();
        } else if (tempSolutionNodes.
                get(tempSolutionNodes.size()-1).
                getCurrCost() < bestSolutionNodes.
                get(bestSolutionNodes.size()-1).
                getCurrCost()) {
            multipleSolution = false;
            searchSolutionNodes.clear();
            for (Node node: bestSolutionNodes){
                if (node.getLevel()<tempSolutionNodes.get(0).getLevel()){
                    searchSolutionNodes.add(node);
                    multipleSolutionNodes.add(node);
                }
            }
            for (Node node: tempSolutionNodes){
                searchSolutionNodes.add(node);
                multipleSolutionNodes.add(node);
            }
            tempSolutionNodes.clear();
        } else { //bobot profit kedua solusi sama
            multipleSolution = true;
            List<Node> solNodesPartition = new ArrayList<>();
            for (Node node : searchSolutionNodes){
                if (node.getLevel() < tempSolutionNodes.get(0).getLevel()){
                    solNodesPartition.add(node);
                }
            }
            for (Node node : tempSolutionNodes){
                solNodesPartition.add(node);
            }
            for (Node node : solNodesPartition){
                multipleSolutionNodes.add(node);
            }
            tempSolutionNodes.clear();
        }
```

Gambar 4.3 Algoritma *branch and bound* bagian 3

```
        }
        //best so far solution cost
        double bestSoFarSolutionCost = this.currNode.getCurrCost();
        activeNodes.remove(currNode);
        List<Node> tempNodes = new ArrayList<>();
        for (Node node : activeNodes){
            tempNodes.add(node);
        }
        for (Node node : tempNodes){
            if (node.getCurrCost() < bestSoFarSolutionCost){ //Bunuh node yang costnya lebih kecil dari best so far solution cost
                activeNodes.remove(node);
            }
        }
        //jika ada node yang costnya lebih besar dari best so far solution cost, ekspansi
        if (!activeNodes.isEmpty()){
            solutionCount += 1;
            this.currNode = activeNodes.get(0);
            tempSolutionNodes.add(currNode);
            i = this.currNode.getLevel();
            level = this.currNode.getLevel();
        }
    }
}
//Luaran kasus solusi tunggal
if (!multipleSolution){
    //Menyusun format luaran hasil berupa "A->B->...->Z"
    String hasil = "";
    int countNode2 = 0;
    for (Node node : bestSolutionNodes){
        countNode2 += 1;
        if (node.getStatus()==1){
            hasil += node.getNama();
            if (countNode2 != bestSolutionNodes.size()){
                hasil += "->";
            }
        }
    }
    //luaran
    System.out.println("Prioritas pengetesan kontak erat konfirmasi positif COVID-19 adalah " + hasil);
    System.out.println("Total waktu aktivitas kontak erat di luar ruangan maksimum adalah " +
            (int) bestSolutionNodes.get(bestSolutionNodes.size()-1).getCurrCost() + " jam");
```

Gambar 4.4 Algoritma *branch and bound* bagian 4

```
    } else { //Luaran kasus solusi jamak
        String hasil = "";
        int countNode2 = 0;
        int solutions = 0;
        for (Node node : multipleSolutionNodes){
            countNode2 += 1;
            if (countNode2 % kontakErats.length == 0){
                solutions += 1;
                System.out.println("Solusi Prioritas pengetesan kontak erat konfirmasi positif COVID-19 ke- "+
                        solutions + " adalah " + hasil);
                hasil = "";
            }
            if (node.getStatus()==1){
                hasil += node.getNama();
                if (countNode2 != multipleSolutionNodes.size()){
                    hasil += "->";
                }
            }
        }

        System.out.println("Total waktu aktivitas kontak erat di luar ruangan maksimum adalah " +
                (int) bestSolutionNodes.get(bestSolutionNodes.size()-1).getCurrCost() + " jam");
    }
}
```

Gambar 4.5 Algoritma *branch and bound* bagian 5

Berikut juga merupakan algoritma fungsi *upper bound* pada TracingBranchBound.java.

```
//Fungsi upper bound / cost
public double upperBound(double currProfit, int kapasitasWaktu, double currWeight, double rasio){
    return (currProfit + ((kapasitasWaktu - currWeight)*rasio));
}
```

Gambar 4.6 Algoritma *branch and bound* bagian fungsi *upper bound*

V. UJI COBA

Uji coba menggunakan kasus uji yang sama seperti pada bab 4. Berikut merupakan prosedur *main program* dengan 6 inputan kontak erat.

```
//Main
public static void main(String args[]){
    KontakErat[] kontakErats = {new KontakErat("A",20,7),new KontakErat("B",15,2)
    ,new KontakErat("C",5,1),new KontakErat("D",15,4),new KontakErat("E",8,3), new KontakErat("F",13,3)};
    TracingBranchBound bestTracing = new TracingBranchBound(kontakErats, 8);
}
```

Gambar 5.1 *main program*

Berikut merupakan luaran dari program.

```
/Users/epatatuah/Library/Java/JavaVirtualMachines/openjdk-16.0.1/Contents/Home/bin/java
Prioritas pengetesan kontak erat konfirmasi positif COVID-19 adalah B->E->F
Total waktu aktivitas kontak erat di luar ruangan maksimum adalah 36 jam

Process finished with exit code 0
```

Gambar 5.2 luaran

Program menghasilkan luaran yang sesuai dengan yang diharapkan, yakni B->E->F dengan total waktu aktivitas maksimum 36 jam.

VI. KESIMPULAN

Konsep algoritma *branch and bound* dapat diterapkan dalam penyelesaian masalah optimasi prioritas pelacakan kontak erat dengan terkonfirmasi positif COVID-19. Penerapan ini dilakukan dengan melakukan pencatatan waktu total si kontak erat dalam melakukan aktivitas di luar rumah dan waktu yang dibutuhkan si *contact tracer* untuk mengunjungi si kontak erat. Dengan menggunakan konsep ini, tidak perlu untuk mengunjungi semua *node* dari pohon status kontak erat. Algoritma *branch and bound* juga menghasilkan nilai yang optimal, hal ini dibuktikan dengan *cost* solusi pertama (35) yang didapat sangat mendekati nilai *cost* solusi optimal (36) namun tetap dilakukan pengecekan hingga berakhir ke nilai *cost* solusi optimal.

VII. UCAPAN TERIMA KASIH

Penulis mengucapkan syukur kepada Tuhan Yang Maha Esa karena berkat dan rahmat-Nya, makalah ini dapat dituntaskan tepat waktu. Penulis berterima kasih kepada Bapak Prof. Ir. Dwi Hendratmo Widyantoro, M.Sc., Ph. D. selaku dosen IF2211 Strategi Algoritma kelas K3 yang telah memberikan ilmu dan juga kesempatan kepada kami untuk mengerjakan tugas makalah ini. Penulis juga berterima kasih kepada semua pihak yang memiliki kontribusi pada pembuatan isi referensi.

LINK VIDEO YOUTUBE

Video penjelasan makalah dapat disaksikan pada pranala berikut: https://youtu.be/8G3QaqFE-YU

REFERENSI

[1] Timurtini, S. 2020. *Virus Corona (COVID-19*). Diakses pada tanggal 10 Mei 2021, dari: https://www.klikdokter.com/penyakit/coronavirus

[2] Ritchie, Hannah. 2020. *Coronavirus (COVID-19) Cases*. Diakses pada tanggal 10 Mei 2021, dari: https://ourworldindata.org/covid-cases

[3] Hamdi, Imam. 2020. *Rumah Sakit Rujukan Overload, Pemerintah Diminta Serius Tekan Penularan COVID-19*. Diakses pada tanggal 10 Mei 2021, dari: https://metro.tempo.co/read/1374996/rumah-sakit-rujukan-overload-pemerintah-diminta-serius-tekan-penularan-covid-19

[4] Widyawati. 2020. *Kemenkes kenalkan istilah Probable, Suspect, Kontak Erat, dan Terkonfirmasi COVID-19*. Diakses pada tanggal 10 Mei 2021, dari: https://sehatnegeriku.kemkes.go.id/baca/umum/20200714/2834469/kemenkes-kenalkan-istilah-probable-suspect-kontak-erat-dan-terkonfirmasi-covid-19/

[5] Kementerian Kesehatan RI. 2020. *Panduan Singkat Pelacakan Kontak (Contact Tracing) untuk Kasus COVID-19*. Diakses pada tanggal 10 Mei 2021, dari: https://infeksiemerging.kemkes.go.id/download/Contact_Tracing_mobile_size_revisi7.pdf

[6] detikcom. 2020. *Satgas COVID-19 Jelaskan Penyebab Ketimpangan Jumlah Testing Antardaerah.* Diakses pada tanggal 10 Mei 2021, dari: https://news.detik.com/berita/d-5164647/satgas-covid-19-jelaskan-penyebab-ketimpangan-jumlah-testing-antardaerahkontak

[7] Munir, Rinaldi. dkk. *Algoritma Branch & Bound*. Diakses pada tanggal 10 Mei 2021, dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2020-2021/Algoritma-Branch-and-Bound-2021-Bagian1.pdf

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, 11 Mei 2021

Epata Tuah/13519120