# BAB III
# ANALISIS DAN PERANCANGAN SISTEM

Pada bab ini akan dibahas mengenai analisis dan perancangan pada sistem yang dibangun yaitu implementasi algoritma *Bee Colony Optimization* (BCO) untuk mencari langkah solusi terpendek pada *puzzle* Rubik's Cube.

## III.1 Deskripsi Masalah

Solusi optimal untuk menyelesaikan *puzzle* Rubik's Cube adalah serangkaian langkah solusi terpendek dari suatu konfigurasi yang acak sampai mencapai konfigurasi yang terselesaikan yaitu warna permukaan *cubies* pada setiap sisi Rubik's Cube menjadi seragam. Solusi tersebut dapat direpresentasikan sebagai jalur terpendek diantara berbagai alternatif jalur yang mungkin dilalui dari suatu keadaan Rubik's Cube yang acak menuju ke keadaan yang terselesaikan pada suatu graf berarah.

Algoritma koloni lebah yang digunakan adalah *Bee Colony Optimization* (*BCO*) dan akan diimplementasikan dalam pencarian jalur terpendek pada suatu simulasi Rubik's Cube.

Untuk mengetahui performansi algoritma *BCO* dalam menyelesaikan *puzzle* Rubik's Cube maka akan dilakukan perbandingan dengan implementasi algoritma lain pada penelitian-penelitian yang telah ada sebelumnya.

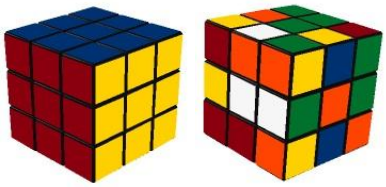
## III.2 Analisis

Analisis merupakan proses identifikasi serta evaluasi mengenai masalah-masalah yang timbul dari pengembangan sistem, baik mengenai aturan serta teknik-teknik yang digunakan dalam pengembangan sistem.

### III.2.1 Analisis Masalah *puzzle* Rubik's Cube

Sebuah Rubik's Cube *standard* memiliki 6 warna yang berbeda pada setiap sisinya yang terdiri atas dua puluh enam *cubies* yang disatukan. Satu *cubies* di tengah dianggap tidak ada karena berperan sebagai sumbu kubus saat memutar sisinya.

Rubik's Cube dimainkan dengan melakukan dengan serangkaian gerakan memutar sisi-sisi Rubik's Cube sedemikian sehingga warna-warna *cubies* pada setiap sisi menjadi teracak. Dari kondisi Rubik's Cube yang acak inilah masalah dimulai, yaitu bagaimana melakukan serangkaian gerakan memutar sisi-sisi Rubik's Cube sedemikian rupa sehingga warna *cubies* pada setiap sisi Rubik's Cube menjadi seragam atau kembali ke keadaan awal. Gambar III.1 memperlihatkan Rubik's Cube dalam kondisi terselesaikan dan acak.



**Gambar III.1 Rubik's Cube dalam kondisi terselesaikan dan acak**

Jika urutan rangkaian gerakan yang telah dilakukan untuk mengacak Rubik's Cube dapat diingat, maka Rubik's Cube dapat dengan mudah dikembalikan ke konfigurasi awal dengan hanya membalikkan urutan rangkaian gerakan yang dilakukan untuk mengacak tadi. Tetapi masalah akan semakin kompleks bila rangkaian gerakan untuk mengacak Rubik's Cube tersebut sangat panjang atau bahkan tidak diketahui sebelumnya, sehingga sulit untuk mengembalikkannya ke konfigurasi awal.

Untuk suatu konfigurasi tertentu terdapat banyak alternatif langkah untuk menyelesaikannya. Penyelesaian *puzzle* Rubik's Cube dengan mengandalkan intuisi maupun dengan langkah-langkah metode tertentu yang dihafalkan manusia biasanya memerlukan lebih dari 40 langkah, sehingga diperlukan teknik optimasi yang tepat untuk menemukan langkah solusi terpendek pada *puzzle* Rubik's Cube

### III.2.1.2 Operator dan Notasi Rubik's Cube

Sebuah Rubik's Cube *standard* memiliki 18 gerakan atau operator yang dapat digunakan untuk mengacak konfigurasinya yaitu:

1. Gerakan memutar 1 dari 6 sisi 90° searah jarum jam. Gambar III.2 memperlihatkan kondisi Rubik's Cube setelah sisi kanan diputar 90° searah jarum jam.



**Gambar III.2 Kondisi setelah operasi 90° searah jarum jam**

2. Gerakan memutar 1 dari 6 sisi 90° berlawanan jarum jam. Gambar III.3 memperlihatkan kondisi Rubik's Cube setelah sisi kanan diputar 90° berlawanan arah jarum jam.



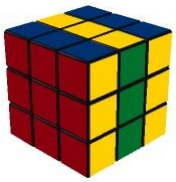
**Gambar III.3 Kondisi setelah operasi 90° berlawanan jarum jam**

3. Gerakan memutar 1 dari 6 sisi 180°. Gerakan ini sama dengan gerakan memutar 1 dari 6 sisi 90° sebanyak dua kali. Gambar III.4 memperlihatkan kondisi Rubik's Cube setelah sisi kanan diputar 180°.



**Gambar III.4 Kondisi setelah operasi 180°**

4. Gerakan memutar bagian tengah Rubik's Cube (*slice*) dianggap sama dengan gerakan memutar dua sisi yang berseberangan dengan arah putaran yang saling berlawanan seperti yang dapat dilihat pada Gambar III.5.



**Gambar III.5 Kondisi setelah operasi *slice* 90°**

Gerakan memutar sisi-sisi atau operator pada Rubik's Cube biasanya dinotasikan dalam notasi Singmaster yaitu U untuk sisi atas (*Up*), D untuk sisi bawah (*Down*), L untuk sisi kiri (*Left*), R untuk sisi kanan (*Right*), F untuk sisi depan (*Front*) dan B untuk sisi belakang (*Back*). Notasi tersebut digunakan untuk menyebutkan operator 90° searah jarum jam, sedangkan untuk putaran berlawanan jarum jam digunakan tanda " ' ". Sebagai contoh, misalnya rangkaian operator " RR'F " berarti memutar sisi kanan sebanyak 90° searah jarum jam, lalu sisi kanan 90° berlawanan jarum jam, dan sisi depan 90° searah jarum jam.

### III.2.1.3 Ruang Keadaan

Ruang keadaan atau total konfigurasi yang mungkin dibangkitkan pada Rubik's Cube untuk digunakan dalam pencarian sangatlah banyak. Pertama (*kemungkinan corner*) yaitu kombinasi setiap *cubies* yang berada di ujung, terdapat 8 kemungkinan tempat yang berbeda dan pada setiap tempat itu satu *cubies* memiliki 3 warna yang berbeda. Kedua (*kemungkinan edge*) yaitu *cubies* yang berada di rusuk, terdapat 12 kemungkinan tempat yang berbeda dan pada setiap tempat itu terdapat 2 warna yang berbeda. Keseluruhan kemungkinan tersebut terdiri dari 12 subgraf yang terpisah tetapi isomorfis (*batasan isomorfis*), sehingga banyaknya konfigurasi yang mungkin adalah :

Kombinasi = (*kemungkinan corner* x *kemungkinan edge) / batasan isomorfis*

$=( (8! \times 3^8) \times (12! \times 2^{12}) ) / 12$

**=43.252.003.274.489.856.000**

$\approx 43 \times 10^{18}$

### III.2.2 Analisis Algoritma *BCO*

Analisis algoritma dilakukan untuk mengetahui optimalitas suatu algoritma yang dapat diukur dari kompleksitas ruang dan kompleksitas waktunya. Algoritma *BCO* akan diterapkan untuk mencari solusi optimal pada Rubik's Cube, yaitu serangkaian langkah terpendek dari konfigurasi yang acak ke konfigurasi yang terselesaikan. Serangkaian langkah tersebut dapat dianggap sebagai jalur terpendek pada suatu graf berarah.

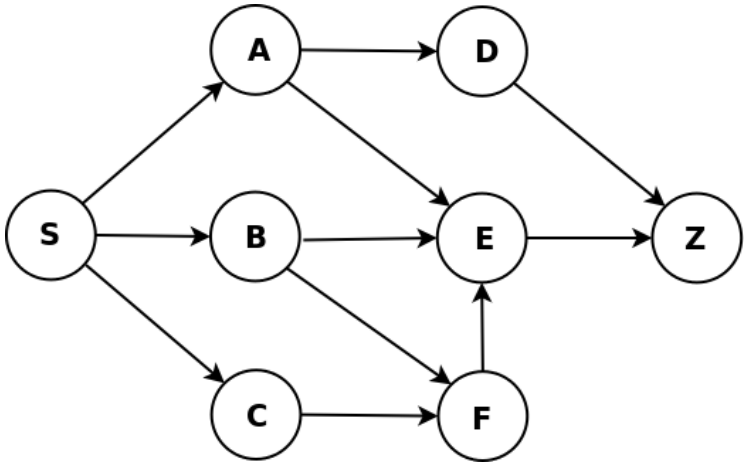
Terdapat beberapa parameter yang harus ditentukan sebelum algoritma BCO dijalankan yaitu jumlah lebah, jumlah iterasi dan jumlah langkah konstruktif. Tahapan-tahapan yang dilakukan dalam pencarian solusi dengan algoritma BCO adalah sebagai berikut :

1. Mengisi posisi setiap lebah dengan simpul awal.
2. *Forward pass*, terdiri dari beberapa langkah yaitu:
   - Lebah melakukan evaluasi terhadap semua simpul yang mungkin dicapai dalam 1 kali langkah konstruktif.
   - Lebah kemudian memilih satu simpul secara acak dengan seleksi *Roulette Wheel* sesuai dengan profitabilitas simpul-simpul tersebut.
3. *Backward pass*, terdiri dari beberapa langkah yaitu
   - Semua lebah akan kembali ke sarang saling membandingkan simpul yang diperolehnya
   - Lebah kemudian memutuskan apakah akan menjadi recruiter dan meneruskan simpul yang diperolehnya, atau menjadi follower dengan mengabaikan simpul yang dipilihnya.
   - Setiap lebah follower kemudian memilih satu simpul baru dari lebah recruiter.

Ketika memasuki tahap *forward pass* ke-dua, terdapat kemungkinan lebih dari satu lebah memilih simpul yang sama sehingga lebah harus dikelompokkan terlebih dahulu sesuai dengan simpulnya, kemudian suksesor dari setiap simpul tersebut akan dievaluasi dan seleksi *roulette wheel* dilakukan secara terpisah untuk setiap simpul yang berbeda.

Tahap *forward pass* ke-dua dan *backward pass* akan diiterasikan sampai simpul tujuan ditemukan, atau jumlah iterasi yang ditentukan sebelumnya tercapai atau dihentikan oleh user.

Untuk mengetahui proses yang terjadi dalam algoritma ini, maka dilakukan pengujian dengan skenario menggunakan suatu graf SABCDEFZ dengan simpul awal S dan simpul tujuan Z pada Gambar III.6.




**Gambar III.6 Graf SABCDEFZ**

Estimasi jarak antara setiap simpul ke simpul tujuan Z telah diketahui sebelumnya diperlihatkan pada Tabel III.1.

**Tabel III.1 Estimasi jarak dari simpul n ke simpul Z**

| **N** | **S** | **A** | **B** | **C** | **D** | **E** | **F** | **Z** |
|---|---|---|---|---|---|---|---|---|
| **h(n)** | 2.5 | 1.20 | 1.5 | 2.5 | 1.0 | 1.0 | 1.5 | 0.0 |

Solusi optimal untuk masalah jalur terpendek pada graf tersebut adalah jalur terpendek di antara semua jalur yang mungkin dilalui dari simpul S ke simpul Z.

Algoritma *BCO* bekerja secara konstruktif dengan membangun solusi-solusi parsial yang ditemukan oleh lebah sampai menjadi suatu solusi yang optimal. Pada graf diatas simpul S dapat dianggap sebagai sarang lebah dan simpul-simpul lainnya dianggap sebagai sumber makanan dengan simpul Z sebagai sumber makanan terbaik.

Misalkan ditentukan jumlah lebah *N* dalam koloni adalah 5 dan gerakan konstruktif yang diperbolehkan adalah 1 langkah untuk setiap iterasi maka, tahapan algoritma *BCO* dalam menyelesaikan masalah jalur terpendek pada graf diatas adalah sebagai berikut:

1. Mengisi posisi setiap lebah dengan simpul awal, Hal ini dapat dilakukan dengan melakukan perulangan untuk setiap lebah dalam koloni sehingga kompleksitas waktu asimptotiknya adalah O(n), dimana n adalah jumlah lebah yang telah ditentukan. Pada kasus ini n=5.
2. Tahap *Forward Pass* pertama

   Setiap lebah mengevaluasi fungsi objektif dari setiap simpul suksesor yaitu simpul-simpul yang mungkin dicapai dari sarang S dalam 1 langkah konstruktif, yaitu simpul A, B dan C. Hal ini dapat dilakukan dengan perulangan untuk setiap simpul suksesor sehingga kompleksitas waktu asimptotiknya adalah O(n), dimana n adalah jumlah simpul suksesor..

   Fungsi objektif atau nilai profitabilitas $Pfi$ suatu simpul pada jalur dapat ditafsirkan sebagai kuantitas nektar yang dikumpulkan oleh lebah ke *i*. Lebah akan mengumpulkan jumlah *nectar* yang lebih banyak bila melakukan perjalanan yang panjang dengan rute lebih pendek. Dengan demikian, $Pfi$ didefinisikan berbanding terbalik dengan panjang tur *Li*, dalam hal ini *Li* adalah *h(n)*. Perhitungan $Pfi$ dilakukan dengan perulangan untuk setiap simpul suksesor sehingga kompleksitas waktu asimptotik adalah O(n), dimana n adalah jumlah simpul suksesor

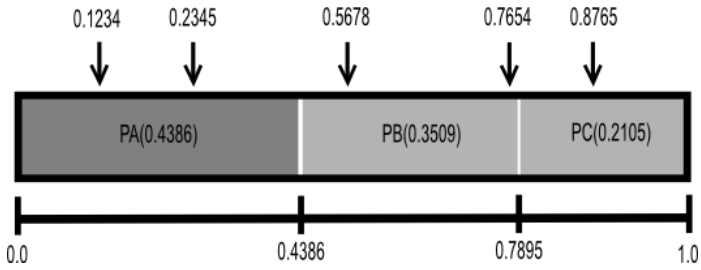
Setiap lebah kemudian memilih satu simpul secara acak dengan seleksi *Roulette Wheel* sesuai dengan fungsi objektifnya.

Hasil perhitungan profitabilitas (*Pfi*) dan probabilitas (*Pi*) dapat dilihat pada Tabel III.2.

**Tabel III.2 Evaluasi Probabilitas Fungsi Objektif Forward Pass pertama**

| Simpul | *Heuristik* | Profitabilitas (*Pfi*) | Probabilitas (*Pi*) |
|---|---|---|---|
| A | 1.20 | 0.8333 | 0.4386 |
| B | 1.50 | 0.6667 | 0.3509 |
| C | 2.5 | 0.40 | 0.2105 |
| | | $\sum_{i=1}^{N} Pfi$ = 1.90 | $\sum_{i=1}^{N} Pi$ = 1 |

*Roulette Wheel* kemudian diisi sesuai dengan nilai *Pi*, dan proses seleksi *Roulette Wheel* dimulai. Perhitungan probabilitas pi dan membuat roulette wheel dilakukan berturut-turut dengan dua perulangan yang terpisah sehingga kompleksitas waktu asimptotik adalah O(n), dimana n adalah jumlah simpul suksesor. Ilustrasi seleksi *Roulette Wheel* dapat dilihat pada Gambar III.7.




**Gambar III.7 Seleksi Roulette Wheel pada tahap Forward Pass pertama**

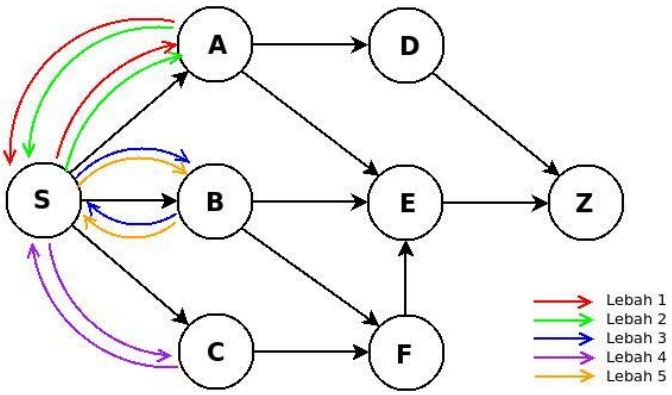
Misalkan dibangkitkan bilangan acak antara 0 sampai 1 sebanyak jumlah lebah *N* berturut turut adalah 0.1234, 0.3456, 0.5678, 0.8765, 0.7654, maka simpul-simpul yang dipilih oleh tiap lebah dapat adalah seperti yang diperlihatkan pada Tabel III.3. Seleksi Roulette Wheel dapat dilakukan dengan melakukan pencarian biner pada Roulette Wheel sesuai bilangan acak yang

dibangkitkan sehingga kompleksitas waktu asimptotiknya adalah O(log n), dimana n adalah jumlah simpul suksesor.

**Tabel III.3 Hasil Roulette Wheel tahap Forward Pass pertama**

| Lebah *i* | Bilangan acak | Simpul yang dipilih |
|---|---|---|
| 1 | 0.1234 | A |
| 2 | 0.2345 | A |
| 3 | 0.5678 | B |
| 4 | 0.8765 | C |
| 5 | 0.7654 | B |

Pergerakan setiap lebah setelah tahap *Forward Pass* pertama dapat dilihat pada Gambar III.8.




**Gambar III.8 Pergerakan lebah setelah tahap Forward Pass pertama**

3. Tahap *Backward Pass*

Pada tahap ini semua lebah kembali ke sarang dan fungsi objektif dari simpul yang dpilih oleh setiap lebah dievaluasi.

Langkah-langkah yang dilakukan dalam tahap *backward pass* adalah sebagai berikut :

Lebah memutuskan secara acak apakah akan menjadi *recruiter* dan meneruskan jalur pencariannya, atau menjadi *follower* dan mengikuti lebah lain. Lebah dengan fungsi objektif lebih baik akan memiliki peluang lebih

besar untuk menjadi *recruiter*. Pada algoritma lebah yang lain langkah ini disebut sebagai *waggle dance*.

Rata rata profitabilitas koloni $Pf_{colony}$ dihitung dengan membagi jumlah total $Pfi$ dalam koloni dengan jumlah lebah *n.*

$Pf_{colony} = (Pf1 + Pf2 + Pf3 + Pf4 + Pf5)/5$

$Pf_{colony} = (0.8333+0.8333+0.6667+0.40+0.6667)/5 = 0.68$

Kompleksitas waktu asimptotik untuk proses menghitung pfcolony adalah O(1).

Probabilitas lebah untuk mengabaikan jalurnya (*Pfollow*) kemudian ditentukan dengan mengikuti aturan pada tabel II.2. (penyesuaian penentuan mengikuti *waggle dance*). Hal ini dilakukan dengan perulangan untuk setiap lebah sehingga kompleksitas waktu asimptotiknya adalah O(n) dimana n adalah jumlah lebah. Tabel III.4 memperlihatkan nilai *Pfollow* dan aturan *Profitability Score* yang diikuti.

**Tabel III.4 Penentuan probabilitas Pfollow**

| **Lebah** | **Simpul** | ***Pfi*** | **Aturan *Profitability Score*** | ***Pfollow*** |
|---|---|---|---|---|
| 1 | A | 0.8333 | **0.90*$Pfcolony \leq Pfi <$ 0.95 * $Pfcolony$**<br>$0.612 \leq Pfi < 0.646$ | 0.20 |
| 2 | A | 0.8333 | **0.90*$Pfcolony \leq Pfi <$ 0.95 * $Pfcolony$**<br>$0.612 \leq Pfi < 0.646$ | 0.20 |
| 3 | B | 0.6667 | **0.90*$Pfcolony \leq Pfi <$ 0.95 * $Pfcolony$**<br>$0.612 \leq Pfi < 0.646$ | 0.20 |
| 4 | C | 0.40 | **$Pfi <$ 0.90*$Pfcolony$**<br>$Pfi < 0.612$ | 0.60 |
| 5 | B | 0.6667 | **0.90*$Pfcolony \leq Pfi <$ 0.95 * $Pfcolony$**<br>$0.612 \leq Pfi < 0.646$ | 0.20 |

Lebah kemudian memutuskan apakah akan menjadi *recruiter* dan meneruskan jalurnya, atau menjadi *follower* dengan mengabaikan jalurnya dan mengikuti lebah lain sesuai dengan probabilitas *Pfollow*.

Misalkan dibangkitkan 5 bilangan acak antara 0 sampai 1 untuk setiap lebah, berturut-turut adalah 0.6543, 0.5678, 0.4321, 0.2345, 0.0123. Semakin kecil nilai *Pfollow*-nya maka lebah akan lebih cenderung menjadi *recruiter*, atau dengan kata lain, bila nilai *Pfollow* kurang dari bilangan acak yang dibangkitkan maka lebah akan menjadi *recruiter* sehingga diperoleh lebah 1(0.20 < 0.6543), lebah 2(0.20<0.5678) dan lebah 3(0.20 < 0.4321) menjadi *recruiter* sedangkan lebah 4(0.60 > 0.2345) dan 5(0.20 > 0.0123) menjadi *follower* dan mengabaikan jalur yang telah diperolehnya. Langkah ini dapat dilakukan dengan melakukan perulangan untuk setiap lebah sehingga kompleksitas waktu asimptotik pada langkah ini adalah O(n), dimana n adalah jumlah lebah. Hasil seleksi *Recruiter-Follower* dapat dilihat pada Tabel III.5.

**Tabel III.5 Hasil seleksi Recuiter-Follower**

| **Lebah *i*** | **Simpul** | ***Pfi*** | **Status** | ***Pfollow*** |
|---|---|---|---|---|
| 1 | A | 0.8333 | *Recruiter* | 0.20 |
| 2 | A | 0.8333 | *Recruiter* | 0.20 |
| 3 | B | 0.6667 | *Recruiter* | 0.20 |
| 4 | - | - | *Follower* | 0.60 |
| 5 | - | - | *Follower* | 0.20 |

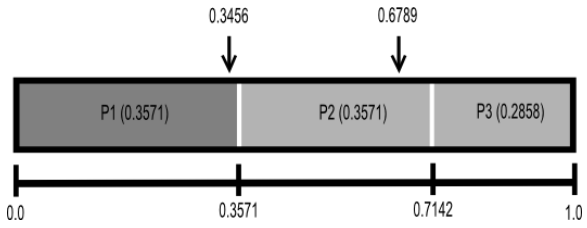
Setiap lebah *follower* memilih jalur baru dari lebah-lebah *recruiter* dengan seleksi *Roulette Wheel.*

Probabilitas *Pi* setiap recruiter dihitung untuk kemudian dimasukkan ke *Roullette Wheel*. Langkah ini dilakukan dengan perulangan sebanyak jumlah lebah *recruiter* sehingga kompleksitas waktu asimptotiknya adalah O(n) dimana n adalah jumlah lebah *recuiter*. Hasil perhitungannya dapat dilihat pada Tabel III.6.

**Tabel III.6 Evaluasi Probabilitas Fungsi Objektif Recruiter**

| **Lebah *i*** | **Simpul** | **Profitabilitas (*Pfi*)** | **Probabilitas (*Pi*)** |
|---|---|---|---|
| 1 | A | 0.8333 | 0.3571 |
| 2 | A | 0.8333 | 0.3571 |
| 3 | B | 0.6667 | 0.2858 |
| | | $\sum_{i=1}^{N} Pfi$ = 2.3333 | $\sum_{i=1}^{N} Pi$ = 1 |

Misalkan dibangkitkan bilangan acak antara 0 sampai 1 sebanyak jumlah *Folower F* berturut turut adalah 0.3456, 0.6789 maka lebah yang akan dikuti oleh lebah 4 adalah lebah 1 (simpul A) sedangkan lebah 5 akan mengikuti lebah 2 (simpul A). Seleksi ini dapat dilakukan dengan pencarian biner pada Roulette wheel yang berisi probabilitas lebah *recruiter* sehingga kompleksitas waktu asimptotik pada tahap ini adalah O(log n) dimana n adalah jumlah lebah *recuiter*. Gambar III.9 memperlihatkan proses seleksi Roullete Wheel pada tahap Backward Pass




**Gambar III.9 Seleksi Roulette Wheel pada tahap Backward Pass**

Tabel III.7 memperlihatkan simpul-simpul yang akan diteruskan oleh setiap lebah setelah tahap *Backward Pass*.

**Tabel III.7 Simpul yang dipilih Lebah setelah Backward Pass**

| **Lebah *i*** | **Bilangan acak** | **Lebah yang diikuti** | **Simpul** |
|---|---|---|---|
| 1 | - | - | A |
| 2 | - | - | A |
| 3 | - | - | B |
| 4 | 0.3456 | 1 | A |
| 5 | 0.6789 | 2 | A |

4. Tahap *Forward Pass* ke-dua

Tahap *Forward Pass* ke-dua pada dasarnya sama dengan tahap *Forward Pass* pertama, hanya saja setiap simpul yang sama hanya akan dievaluasi satu kali.

Lebah-lebah dengan simpul yang sama dikelompokkan, yaitu lebah 1,2,4 dan 5 pada simpul A, dan lebah 3 pada simpul B. Langkah ini dilakukan dengan melakukan perulangan bersarang untuk membandingkan simpul-simpul yang dipilih oleh lebah *recuiter* sehingga kompleksitas waktu asimptotiknya adalah $O(n^2)$ dimana n adalah jumlah *recuiter*.

Profitabilitas setiap simpul suksesor yang mungkin dicapai dari setiap simpul yang berbeda (*distinct nodes*) dievaluasi. Kompleksitas waktu asimptotik pada langkah ini adalah O(n), dimana n adalah jumlah simpul suksesor.

Roulette Wheel kemudian dilakukan terpisah untuk masing-masing kelompok seperti yang diperlihatkan pada Tabel III.8.

**Tabel III.8 Evaluasi probabilitas fungsi objektif Forward Pass ke-dua**

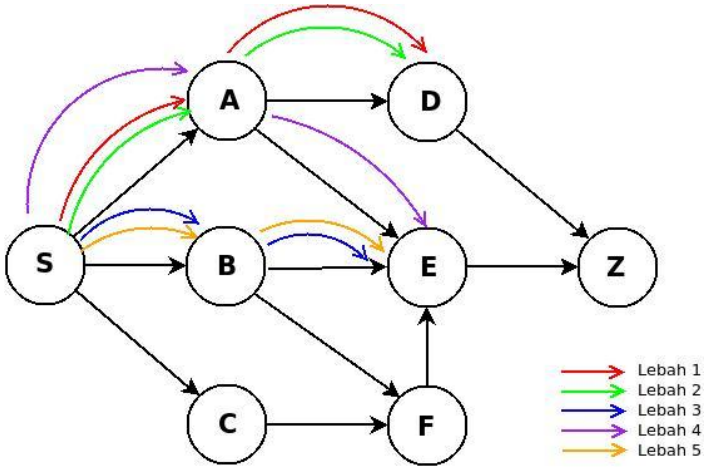
| **Simpul *i*** | **Lebah** | **Simpul Berikutnya** | ***Li*** | ***Pfi*** | ***Pi*** |
|---|---|---|---|---|---|
| A | 1,2,4,5 | D | 1.0 | 1 | 0.50 |
| | | E | 1.0 | 1 | 0.50 |
| | | | | $\sum_{i=1}^{N} Pfi = 2$ | $\sum_{i=1}^{N} Pi = 1$ |
| B | 3 | E | 1.0 | 1 | 0.60 |
| | | F | 1.5 | 0.6667 | 0.40 |
| | | | | $\sum_{i=1}^{N} Pfi = 1.6667$ | $\sum_{i=1}^{N} Pi = 1$ |

Misalkan dibangkitkan bilangan acak berturut-turut 0.1234, 0.2345, 0.5432, 0.6543 untuk 4 lebah yang memilih simpul A, dan *Roulette Wheel* berisi 0.0 sampai 0.50 untuk simpul D dan 0.50 sampai 1 untuk simpul E, maka simpul yang akan dipilih oleh lebah 1 dan 2 adalah simpul D, sedangkan lebah 4 dan 5 memilih simpul E.

Misalkan dibangkitkan bilangan acak untuk lebah 3 yang memilih simpul B adalah 0.5678, dan *Roulette Wheel* berisi 0.0 sampai 0.60 untuk simpul E dan 0.60 sampai 1 untuk simpul F maka lebah 3 akan memilih simpul E.

Karena tahap ini dilakukan untuk setiap simpul berbeda maka kompleksitas waktu asimptotiknya adalah $O(n^2)$ dimana n adalah jumlah simpul yang berbeda.

Posisi lebah setelah tahap *Forward Pass* ke-dua dapat dilihat pada Gambar III.10.




**Gambar III.10 Posisi lebah setelah tahap Forward Pass ke-dua**

5. Tahap *Backward Pass* ke-dua

Tahap *Backward Pass* sama seperti tahap *Backward Pass* pertama, dan selanjutnya ke-dua tahap (tahap *Backward Pass* pertama dan *Forward Pass* ke-dua) diiterasikan sampai ada lebah yang menemukan simpul tujuan.

Karena $Pfi$ semua simpul yang dipilih oleh lebah sama dengan 1 maka $Pf_{colony}$= 1, dan aturan yang digunakan untuk menentukan pfollow adalah ($0.95 * Pf_{colony} \leq Pfi < 1.15 * Pf_{colony}$) dengan nilai pfollow 0.02.

Misalkan dibangkitkan 5 bilangan acak antara 0 sampai 1 untuk setiap lebah, berturut-turut adalah 0.6543, 0.5678, 0.4321, 0.2345, 0.0123, maka lebah 1, 2, 3, dan 4 menjadi *recruiter*, sedangkan lebah 5 menjadi *follower*.

Langkah 2

Karena $Pfi$ lebah 1, 2, 3 dan 4 bernilai sama yaitu 1, maka probabilitas untuk roulette wheel adalah seperempat atau 0.25 untuk masing-masing *recruiter*. Misalkan dibangkitkan bilangan acak untuk lebah 5 adalah 0.4321, maka lebah 5 akan mengikuti lebah ke 2, sehingga posisi terakhir setiap lebah adalah lebah 1,2 dan 5 memilih simpul D, lebah 3 dan 4 memilih simpul E

6. Tahap *Forward Pass* ke-tiga

Pada tahap ini kedua *distinct node* (simpul D dan E) memiliki suksesor yang sama yaitu simpul tujuan Z. Misalkan kelompok pertama yang memilih

simpul D diseleksi terlebih dahulu maka lebah 1 yang menjadi lebah pertama yang menemukan simpul tujuan, dan iterasi algoritma *BCO* berakhir.

Dari skenario diatas kompleksitas waktu asimptotik algoritma BCO dalam pencarian jarak terpendek dapat dirangkum pada Tabel III.9.

**Tabel III.9 Perhitungan Kompleksitas Waktu Asimptotik**

| Tahap | Langkah | Big Oh |
|---|---|---|
| Tahap awal | Mengisi lebah dengan simpul awal | O(n) |
| Forward Pass pertama | Evaluasi fungsi objektif | O(n) |
| | Perhitungan *Pfi* | O(n) |
| | Perhitungan *Pi* | O(n) |
| | Pembuatan Roulette | O(n) |
| | Seleksi Roulette Wheel | O(log n) |
| Total | | O(n) |
| Backward Pass | Perhitungan *Pfcolony* | O(1) |
| | Penentuan *Pfollow* | O(n) |
| | Seleksi Recruiter/Follower | O(n) |
| | Perhitungan *Pi* Recruiter | O(n) |
| | Roulette Wheel Follower ke Recruiter | O(log n) |
| Total | | O(n) |
| Forward Pass kedua | Mengelompokkan Recruiter | $O(n^2)$ |
| | Perhitungan *Pfi* | O(n) |
| | Perhitungan *Pi* | O(n) |
| | Roulette setiap kelompok | $O(n^2)$ |
| Total | | $O(n^2)$ |

Kompleksitas waktu asimptotik keseluruhan tahap adalah:

$O(n)+O(n)+O(n)+O(n^2) = O(n^2)$

Akan tetapi tahap backward Pass dan Forward Pass kedua akan diiterasikan sampai solusi ditemukan. Bila diasumsikan solusi ditemukan pada kedalaman m maka kedua tahap tersebut akan diiterasikan sebanyak m kali sehingga kompleksitas waktu asimptotiknya menjadi $O(n^2).m = O(n^3)$.

### III.2.3 Analisis Struktur Data

Analisis struktur data digunakan untuk merancang struktur data yang akan digunakan dalam implementasi algoritma.

1. Simpul

   Struktur data untuk menyimpan simpul adalah *singly linked list*, dimana setiap simpul akan menunjuk pada simpul *parent*.

2. Representasi Graf

   Solusi optimal untuk suatu konfigurasi Rubik's Cube dapat dipandang sebagai jalur terpendek dari konfigurasi tersebut menuju ke konfigurasi yang terselesaikan pada sebuah graf berarah. Jumlah konfigurasi yang mungkin dihasilkan pada *puzzle* Rubik's Cube sangat banyak sehingga bila graf direpresentasikan secara eksplisit akan membutuhkan memori yang sangat besar untuk menyimpan simpul-simpulnya, baik dengan menggunakan matriks kedekatan maupun dengan senarai kedekatan.

   Sebagai alternatif graf tersebut dapat dibangun secara implisit, dimana setiap simpul dalam graf hanya akan menyimpan informasi mengenai simpul asalnya (*parent*) dan operator yang membangkitkannya (*edge*), sedangkan suksesor untuk setiap simpul dibangkitkan dan disimpan dalam array dinamis berdimensi satu dan diseleksi secara lokal dalam subrutin algoritma.

3. Koloni Lebah

   Jumlah lebah dalam koloni setiap kali algoritma dijalankan adalah tetap sehingga dapat disimpan pada sebuah array statis berdimensi satu. Lebah akan dipisahkan antara lebah *follower* dan lebah *recruiter* pada tahap *Backward Pass*, masing-masing disimpan dalam dua array dinamis berdimensi satu.

### III.2.4 Analisis Simulasi Rubik's Cube

Simulator yang digunakan untuk memasukkan konfigurasi dan memvisualisasikan *puzzle* Rubik's Cube diadaptasi dari Rubik's Cube Java Applet versi 1.3.1 oleh Michael Schubart yang dimodifikasi oleh Uday K. Reddy B. untuk solver dengan IDA*.

Kelas-kelas yang digunakan dari applet ini adalah

1. Kelas Cubelet untuk mendefinisikan 27 *cubies* yang membentuk sebuah Rubik's Cube. Sebuah *cubies* terdiri dari 6 sisi yang direpresentasikan dalam sebuah Array berisi 6 warna sisi yang berbeda yaitu biru, putih, oranye, merah, kuning dan hijau. Tiap warna direpresentasikan dalam 1 byte bernilai 0 sampai 5 yang mewakili 6 warna tersebut.

   Metode-metode yang disediakan pada kelas ini adalah rotate() digunakan untuk mengubah orientasi *cubies,* getColor() digunakan untuk mendapatkan salah satu warna sisi cubies.
2. Kelas Cube untuk mendefinisikan sebuah Rubik's Cube utuh yang terdiri dari 27 *cubies.* Ruang memori yang dibutuhkan untuk menyimpan sebuah Cube membutuhkan 27 kali 6 byte yaitu 162 byte.

   Metode-metode yang disediakan pada kelas ini adalah copy() digunakan untuk menduplikasi suatu konfigurasi Cube, getColor() digunakan untuk mendapatkan warna salah satu cubies, isSolved() untuk memeriksa apakah konfigurasi Rubik's Cube dalam kondisi terselesaikan, dan rotate() untuk memutar sisi-sisi Rubik's Cube.
3. Kelas CubeSurfacePoint untuk mendefinisikan titik pada permukaan Rubik's Cube.

   Metode-metode yang disediakan kelas ini adalah isOnCube() untuk memeriksa apakah titik yang ditunjuk oleh mouse terdapat pada permukaan Rubik's Cube dan transform() untuk mengubah titik yang ditunjuk oleh mouse menjadi titik CubeSurfacePoint.
4. Kelas CubeDisplayEntry sebagai *control* untuk mendefinisikan elemen-elemen Rubik's Cube yang akan di tampilkan ke layar.

5. Kelas Heuristics untuk menentukan jarak 3D Manhattan Distance dari suatu konfigurasi Rubik's Cube.

   Metode-metode yang disediakan adalah faceinit() dan initpos() untuk menginisialisasi sisi-sisi dan posisi setiap cubies yang menyusun Rubik's Cube, getCorrectPos() untuk mendapatkan posisi dan orientasi sebenarnya dari suatu *cubies* dan heur() untuk menghitung jarak 3D Manhattan Distance dari suatu konfigurasi Rubik's Cube.
6. Kelas RubiksCubeApplet untuk menampilkan visualisasi Rubik's Cube. Kelas ini mengimplementasikan Applet sehingga pada penerapannya nanti, kelas ini harus dimodifikasi agar dapat berjalan sebagai aplikasi *standalone*, yaitu dengan menjadikannya sebagai *subclass* untuk kemudian ditampilkan pada sebuah jPane. Metode-metode yang disediakan adalah getCube() untuk mendapatkan konfigurasi Rubik, drawFace() untuk menggambar konfigurasi Rubik, setDragStart() dan getDragStart() untuk menentukan dan membaca koordinat yang ditunjuk dengan *mouse.*

### III.2.5 Analisis Pemecahan Masalah Dengan Algoritma *BCO*

Solusi untuk menyelesaikan *puzzle* Rubik's Cube adalah serangkaian langkah atau gerakan memutar sisi-sisi Rubik's Cube sedemikian rupa sehingga semua sisi memiliki warna yang seragam. Solusi ini dapat disebut optimal bila solusi tersebut merupakan serangkaian langkah terpendek yang mungkin dilakukan untuk menyelesaikan suatu konfigurasi Rubik's Cube.

Algoritma BCO yang diusulkan dapat dibagi menjadi 2 tahap utama yang harus dilakukan yaitu *forward pass* dan *backward pass.* Beberapa hal yang perlu diperhatikan sebelum algoritma dijalankan adalah:

a. Heuristik

   Fungsi objektif yang digunakan oleh lebah dalam proses pengambilan keputusan adalah *3D Manhattan Distance*. Fungsi heuristik ini telah sering dan berhasil digunakan pada beberapa metode lain dalam menyelesaikan Rubik's Cube.

b. Seleksi *Roulette Wheel*

Seleksi *Roulette Wheel* pada algoritma *BCO* bekerja sedemikian sehingga simpul-simpul dengan fungsi objektif yang lebih besar mendapat peluang lebih besar untuk dipilih. Pada kasus Rubik's Cube, semakin kecil *Manhattan Distance* suatu simpul, semakin baik pula fungsi objektifnya, maka agar dapat digunakan pada seleksi *Roulette Wheel*, nilai fungsi tersebut harus diinverskan terlebih dahulu sehingga semakin kecil fungsi heuristiknya maka kualitas simpul yang akan dipilih semakin baik.

c. Gerakan konstruktif dan operator

Gerakan konstruktif yang dilakukan oleh lebah dibatasi 1 langkah pada setiap iterasi. Algoritma *BCO* bekerja sedemikian sehingga semua lebah berada pada kedalaman yang sama dalam setiap iterasinya maka dengan mengabaikan 6 operator 180° yang dapat menyebabkan lebah berada pada kedalaman yang berbeda, jumlah operator yang diijinkan adalah 12 yaitu U, U', D, D', F, F', R, R', L, L', B, dan B'.

Misalkan diketahui suatu konfigurasi Rubik's Cube frb yang diperoleh dengan operator FRB pada Gambar III.11.



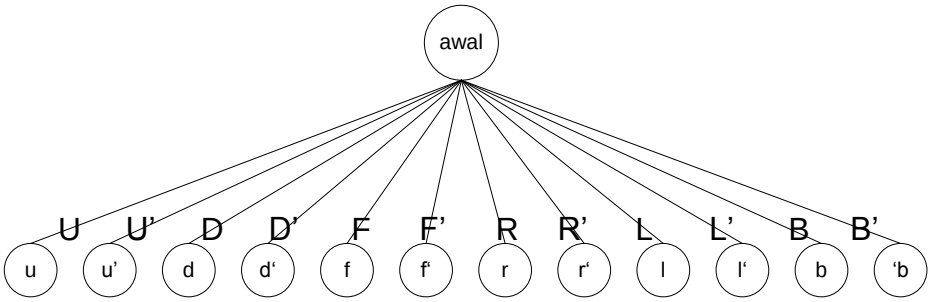
**Gambar III.11 Konfigurasi Rubik FRB**

Kemudian ditentukan parameter jumlah lebah 12, maka tahapan algoritma *BCO* untuk menyelesaikan kasus diatas adalah sebagai berikut.

1. Posisi setiap lebah diisi dengan simpul awal yang berisi konfigurasi frb.
2. Tahap *forward pass* pertama

Setiap lebah mengevaluasi fungsi objektif dari 12 simpul suksesor yang mungkin dicapai dari simpul awal dalam 1 langkah konstruktif, yaitu simpul u, u', d, d', f, f', r, r', l, l', b dan b' yang berisi konfigurasi yang

diperoleh dari operator 12 operator yang diperbolehkan (U, U', D, D', F, F', R, R', L, L', B, dan B'). Gambar III.12 memperlihatkan simpul awal dengan 12 simpul suksesornya.




**Gambar III.12 Simpul Awal dengan 12 Suksesor**

Nilai profitabilitas (*Pfi)* setiap simpul suksesor dievaluasi kemudian probabilitasnya (*Pi*) dihitung. Hasil perhitungan *Pfi* dan *Pi* dapat dilihat pada Tabel III.10.

**Tabel III.10 Evaluasi Nilai Pfi dan Pi**

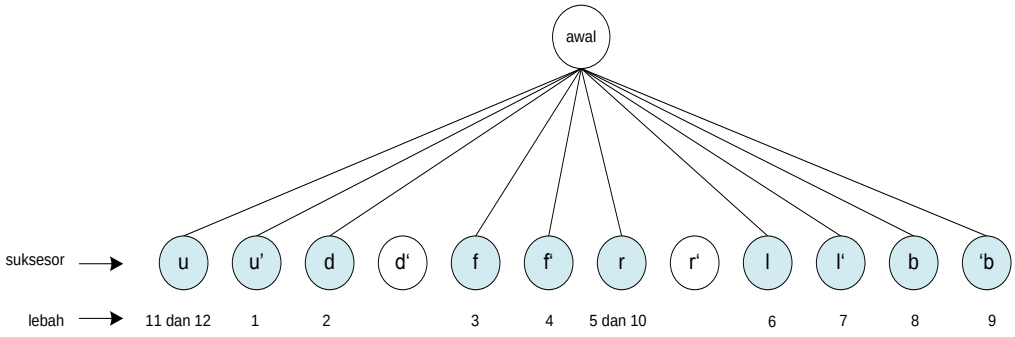
| **Simpul** | **Operator** | ***f*** | ***Pfi*** | ***Pi*** |
|---|---|---|---|---|
| U | U | 6.5 | 0.1538 | 0.0726 |
| u' | U' | 5.75 | 0.1739 | 0.0821 |
| D | D | 5 | 0.2000 | 0.0944 |
| d' | D' | 6.75 | 0.1481 | 0.0699 |
| F | F | 6.5 | 0.1538 | 0.0726 |
| f' | F' | 4.5 | 0.2222 | 0.1049 |
| R | R | 6.5 | 0.1538 | 0.0726 |
| r' | R' | 5.5 | 0.1818 | 0.0858 |
| L | L | 5 | 0.2 | 0.0944 |
| l' | L' | 6.5 | 0.1538 | 0.0726 |
| B | B | 6.5 | 0.1538 | 0.0726 |
| b' | B' | 4.5 | 0.2222 | 0.1049 |
| | | | $\sum_{i=1}^{N} Pfi$= 2.1175 | $\sum_{i=1}^{N} Pi$ = 1 |

*Roulette Wheel* kemudian diisi sesuai dengan nilai *Pi*, kemudian seleksi *Roulette Wheel* dilakukan dengan membangkitkan 12 bilangan acak masing-masing untuk setiap lebah, kemudian setiap lebah diberikan simpul sesuai dengan bilangan acaknya. Tabel III.11 memperlihatkan hasil seleksi *Roulette Wheel*.

**Tabel III.11 Hasil tahap forward pass pertama**

| Simpul | Roulette Wheel | Lebah | Bilangan acak | Simpul yang dipilih |
|---|---|---|---|---|
| u | 0 - 0.0726 | 1 | 0.1234 | u' |
| u’ | 0.0726 - 0.1547 | 2 | 0.2345 | D |
| d | 0.1547 - 0.2492 | 3 | 0.3456 | F |
| d’ | 0.2492 - 0.3191 | 4 | 0.4567 | f' |
| f | 0.3191 - 0.3918 | 5 | 0.5678 | R |
| f’ | 0.3918 - 0.4967 | 6 | 0.6789 | L |
| r | 0.4967 - 0.5694 | 7 | 0.7654 | l' |
| r’ | 0.5694 - 0.6553 | 8 | 0.8765 | B |
| l | 0.6553 - 0.7497 | 9 | 0.9876 | b' |
| l’ | 0.7497 - 0.8224 | 10 | 0.5432 | R |
| b | 0.8224 - 0.8950 | 11 | 0.0234 | U |
| b’ | 0.8950 - 1 | 12 | 0.0123 | U |

Posisi setiap lebah setelah tahap *forward pass* pertama dapat dilihat pada Gambar III.13.




**Gambar III.13 Posisi Lebah setelah Forward Pass pertama**

3. Tahap Backward Pass

Pada tahap ini semua lebah kembali ke sarang dan fungsi objektif dari simpul yang dpilih oleh setiap lebah dievaluasi.

Kemudian lebah memutuskan secara acak apakah akan menjadi *recruiter* dan meneruskan jalur pencariannya, atau menjadi *follower* dan mengikuti lebah lain. Rata rata profitabilitas koloni $Pf_{colony}$ dihitung dengan membagi jumlah total *Pfi* dalam koloni dengan jumlah lebah *n.* $Pf_{colony}$ dihitung dengan persamaan (2.3) sebagai berikut:

$Pf_{colony}$ = 2.117554587/12

$Pf_{colony}$ = 0.176462882

Probabilitas lebah untuk mengabaikan jalurnya (*Pfollow*) kemudian ditentukan dengan mengikuti aturan pada tabel II.2. Hasil perhitungannya dapat dilihat pada Tabel III.12.

**Tabel III.12 Hasil penentuan Probabilitas Pfollow**

| **Lebah** | **Bilangan acak** | **Simpul yang dipilih** | ***Pfi*** | **Aturan *Profitability Score*** | ***Pfollow*** |
|---|---|---|---|---|---|
| 1 | 0.1234 | u' | 0.1739 | 0.95*Pfcolony ≤ Pfi < 1.15 * Pfcolony | 0.02 |
| 2 | 0.2345 | d | 0.2 | 0.95*Pfcolony ≤ Pfi < 1.15 * Pfcolony | 0.02 |
| 3 | 0.3456 | f | 0.1538 | Pfi < 0.90*Pfcolony | 0.6 |
| 4 | 0.4567 | f' | 0.2222 | 1.15*Pfcolony ≤ Pfi | 0 |
| 5 | 0.5678 | r | 0.1538 | Pfi < 0.90*Pfcolony | 0.6 |
| 6 | 0.6789 | l | 0.2 | 0.95*Pfcolony ≤ Pfi < 1.15 * Pfcolony | 0.02 |
| 7 | 0.7654 | l' | 0.1538 | Pfi < 0.90*Pfcolony | 0.6 |
| 8 | 0.8765 | b | 0.1538 | Pfi < 0.90*Pfcolony | 0.6 |
| 9 | 0.9876 | b' | 0.2222 | 1.15*Pfcolony ≤ Pfi | 0 |
| 10 | 0.5432 | r | 0.1538 | Pfi < 0.90*Pfcolony | 0.6 |
| 11 | 0.0234 | u | 0.1538 | Pfi < 0.90*Pfcolony | 0.6 |
| 12 | 0.0123 | u | 0.1538 | Pfi < 0.90*Pfcolony | 0.6 |

Lebah kemudian memutuskan apakah akan menjadi *recruiter* dan meneruskan jalurnya, atau menjadi *follower* dengan mengabaikan jalurnya dan

mengikuti lebah lain sesuai dengan probabilitas *Pfollow*. Langkah ini dilakukan dengan membangkitkan 12 bilangan acak untuk masing-masing lebah, bila bilangan acak kurang dari nilai *Pfollow* maka lebah akan menjadi *recuiter* dan meneruskan simpulnya, bila tidak maka lebah akan menjadi *follower* yaitu mengabaikan simpulnya dan mengikuti lebah *recuiter*. Hasil seleksi *recruiter* atau *follower* dapat dilihat pada Tabel III.13.

**Tabel III.13 Hasil seleksi Recruiter atau Follower**

| Lebah | Simpul | *Pfi* | Bilangan Acak | *Pfollow* | Status |
|---|---|---|---|---|---|
| 1 | u' | 0.1739 | 0.2345 | 0.02 | Recruiter |
| 2 | d | 0.2 | 0.8765 | 0.02 | Recruiter |
| 3 | - | 0.1538 | 0.0654 | 0.6 | Follower |
| 4 | f' | 0.2222 | 0.4567 | 0 | Recruiter |
| 5 | - | 0.1538 | 0.5678 | 0.6 | Follower |
| 6 | l | 0.2 | 0.6789 | 0.02 | Recruiter |
| 7 | l' | 0.1538 | 0.7654 | 0.6 | Recruiter |
| 8 | b | 0.1538 | 0.6543 | 0.6 | Recruiter |
| 9 | b' | 0.2222 | 0.1234 | 0 | Recruiter |
| 10 | - | 0.1538 | 0.5432 | 0.6 | Follower |
| 11 | - | 0.1538 | 0.0234 | 0.6 | Follower |
| 12 | - | 0.1538 | 0.2345 | 0.6 | Follower |

Probabilitas *Pi* setiap lebah *recuiter* dihitung berdasarkan profitabilitasnya kemudian dimasukkan ke *Roulette Wheel*. Kemudian setiap lebah *follower* memilih jalur baru dari lebah-lebah *recruiter* dengan seleksi *Roulette Wheel. Roulette Wheel* lebah *follower* ke lebah *recuiter* dapat dilihat pada pada Tabel III.14.

**Tabel III.14 Roulette follower ke recruiter**

| Lebah *Recruiter* | Simpul | *Pfi* | *Pi* | Roulete Wheel |
|---|---|---|---|---|
| 1 | u' | 0.1739 | 0.1311 | 0 - 0.1311 |
| 2 | d | 0.2 | 0.1508 | 0.1311 - 0.2819 |
| 4 | f' | 0.2222 | 0.1675 | 0.2819 - 0.4495 |
| 6 | l | 0.2 | 0.1508 | 0.4495 - 0.6003 |
| 7 | l' | 0.1538 | 0.1160 | 0.6003 - 0.7163 |
| 8 | b | 0.1538 | 0.1160 | 0.7163 - 0.8324 |
| 9 | b' | 0.2222 | 0.1675 | 0.8324 - 1 |

Seleksi *Roulette Wheel* dilakukan dengan membangkitkan 5 bilangan acak untuk 5 lebah *follower*, kemudian setiap lebah *follower* diberikan simpul lebah *recuiter* sesuai dengan bilangan acaknya. Hasil seleksi Roulette Wheel *follower* ke *recuiter* dapat dilihat pada Tabel III.15.

**Tabel III.15 Hasil Roulette Wheel follower ke recruiter**

| Lebah *Follower* | Bilangan acak | Lebah *Recruiter* yang diikuti | Simpul yang dipilih |
|---|---|---|---|
| 3 | 0.5432 | 6 | l |
| 5 | 0.0234 | 1 | u' |
| 10 | 0.3456 | 4 | f' |
| 11 | 0.4567 | 6 | l |
| 12 | 0.8765 | 9 | b' |

4. Tahap forward pass ke 2

Tahap *Forward Pass* ke-dua pada dasarnya sama dengan tahap *Forward Pass* pertama, hanya saja setiap simpul yang sama hanya akan dievaluasi satu kali.

Lebah-lebah dengan simpul yang sama dikelompokkan dengan membandingkan simpul-simpul yang dipilih oleh lebah recruiter. Untuk setiap simpul yang berbeda (*distinct nodes*), evaluasi 11 suksesor yang tidak menuju ke simpul parent.

Roulette Wheel yang dibangun untuk simpul u' yang dipilih oleh lebah 1 dan 5 dapat dilihat pada Tabel III.16.

**Tabel III.16 Roulette Wheel Untuk Simpul u'**

| Simpul | Lebah | Operator - Simpul | *f* | *Pfi* | *Pi* | Roulette |
|---|---|---|---|---|---|---|
| u' | 1 dan 5 | U – u' | - | - | - | - |
| | | U' – u'u' | 7 | 0.1428 | 0.0856 | 0 - 0.0856 |
| | | D – u'd | 6 | 0.1666 | 0.0999 | 0.0856 - 0.1856 |
| | | D' – u'd' | 9 | 0.1111 | 0.0666 | 0.1856 - 0.2522 |
| | | F – u'f | 6.5 | 0.1538 | 0.0922 | 0.2522 - 0.3444 |
| | | F' – u'f' | 6.5 | 0.1538 | 0.0922 | 0.3444 - 0.4367 |
| | | R – u'r | 6.5 | 0.1538 | 0.0922 | 0.4367 - 0.5290 |
| | | R' – u'r' | 5.75 | 0.1739 | 0.1042 | 0.5290 - 0.6332 |
| | | L – u'l | 6.5 | 0.1538 | 0.0922 | 0.6332 - 0.7255 |
| | | L' – u'l' | 7 | 0.1428 | 0.0856 | 0.7255 - 0.8112 |
| | | B – u'b | 6.75 | 0.1481 | 0.0888 | 0.8112 - 0.9000 |
| | | B' – u'b' | 6 | 0.1666 | 0.0999 | 0.9000 - 1 |
| | | | sum | 1.6676 | 1 | |

Kemudian *Roulette Wheel* dilakukan secara terpisah untuk setiap simpul yang berbeda tersebut. Tabel III.17 memperlihatkan hasil seleksi *Roulette Wheel* untuk simpul u' yang dipilih oleh lebah 1 dan 5.

**Tabel III.17 Hasil seleksi Roulette Wheel simpul u'**

| Lebah | Bilangan acak | next op | simpul |
|---|---|---|---|
| 1 | 0.3456 | F' | u'f' |
| 5 | 0.8765 | B | u'b |

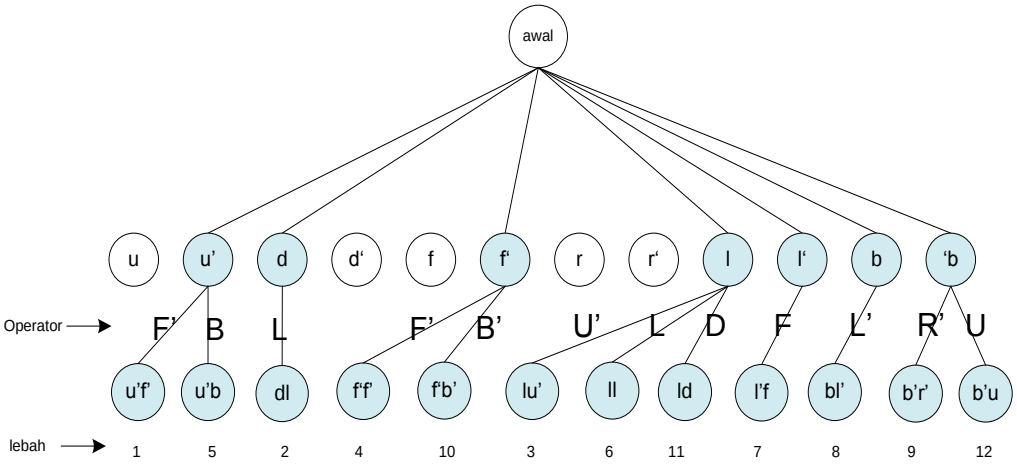
Pada kasus ini, *Roulette Wheel* dilakukan sebanyak 7 kali yaitu jumlah recruiter dengan simpul yang berbeda,

Hasil keseluruhan *Roulette Wheel* untuk ke 7 simpul tersebut dapat dilihat pada Tabel III.18.

**Tabel III.18 Hasil tahap forward pass ke-dua**

| **simpul parent** | **lebah** | **operator** | **simpul yang dipilih** | ***Pfi*** |
|---|---|---|---|---|
| u' | 1 | F' | u'f' | 0.1538 |
| | 5 | B | u'b | 0.1481 |
| d | 2 | L | dl | 0.2222 |
| f' | 4 | F' | f'f' | 0.1538 |
| | 10 | B' | f'b' | 0.2857 |
| l | 3 | U' | lu' | 0.1666 |
| | 6 | L | ll | 0.1666 |
| | 11 | D | ld | 0.2222 |
| l' | 7 | F | l'f | 0.1538 |
| b | 8 | L' | bl' | 0.1428 |
| b' | 9 | R' | b'r' | 0.3333 |
| | 12 | U | b'u | 0.1818 |

Posisi setiap lebah setelah *forward pass* ke-dua diperlihatkan pada Gambar III.14.




**Gambar III.14 Posisi lebah setelah forward pass ke-dua**

5. Tahap *Backward Pass* ke-dua

Tahap ini sama dengan tahap *backward pass* pertama yaitu, semua lebah kembali ke sarang dan fungsi objektif dari simpul yang dpilih oleh setiap lebah dievaluasi.

$Pf_{colony}$ dihitung dengan persamaan (2.3) sebagai berikut:

$Pf_{colony}$ = 2.331187331/12

$Pf_{colony}$ = 0.194266

Probabilitas lebah untuk mengabaikan jalurnya (*Pfollow*) kemudian ditentukan dengan mengikuti aturan pada Tabel II.2. Hasil perhitungannya dapat dilihat pada Tabel III.19.

**Tabel III.19 Hasil penentuan probabilitas Pfollow tahap backward pass ke-dua**

| **Lebah** | **Bilangan acak** | **Simpul yang dipilih** | ***Pfi*** | **Aturan *Profitability Score*** | ***Pfollow*** |
|---|---|---|---|---|---|
| 1 | 0.1234 | u'f' | 0.1538 | Pfi < 0.90*Pfcolony | 0.6 |
| 2 | 0.2345 | Dl | 0.2222 | 0.95*Pfcolony ≤ Pfi < 1.15 * Pfcolony | 0.02 |
| 3 | 0.3456 | lu' | 0.1666 | Pfi < 0.90*Pfcolony | 0.6 |
| 4 | 0.4567 | f'f' | 0.1538 | Pfi < 0.90*Pfcolony | 0.6 |
| 5 | 0.5678 | u'b | 0.1481 | Pfi < 0.90*Pfcolony | 0.6 |
| 6 | 0.6789 | Ll | 0.1666 | Pfi < 0.90*Pfcolony | 0.6 |
| 7 | 0.7654 | l'f | 0.1538 | Pfi < 0.90*Pfcolony | 0.6 |
| 8 | 0.8765 | bl' | 0.1428 | Pfi < 0.90*Pfcolony | 0.6 |
| 9 | 0.9876 | b'r' | 0.3333 | 1.15*Pfcolony ≤ Pfi | 0 |
| 10 | 0.5432 | f'b' | 0.2857 | 1.15*Pfcolony ≤ Pfi | 0 |
| 11 | 0.0234 | Ld | 0.2222 | 0.95*Pfcolony ≤ Pfi < 1.15 * Pfcolony | 0.02 |
| 12 | 0.0123 | b'u | 0.1818 | 0.90*Pfcolony ≤ Pfi < 0.95 * Pfcolony | 0.2 |

Lebah kemudian memutuskan apakah akan menjadi *recruiter* dan meneruskan jalurnya, atau menjadi *follower* dengan mengabaikan jalurnya dan

mengikuti lebah lain sesuai dengan probabilitas *Pfollow*. Hasil seleksi recruiter atau follower diperlihatkan pada Tabel III.20.

**Tabel III.20 Hasil seleksi recruiter-follower tahap backward pass ke-dua**

| Lebah | Simpul | *Pfi* | Bilangan Acak | *Pfollow* | Status |
|---|---|---|---|---|---|
| 1 | u'f' | 0.1538 | 0.2345 | 0.02 | Recruiter |
| 2 | dl | 0.2222 | 0.8765 | 0.02 | Recruiter |
| 3 | - | 0.1666 | 0.0654 | 0.6 | Follower |
| 4 | f'f' | 0.1538 | 0.4567 | 0 | Recruiter |
| 5 | - | 0.1481 | 0.5678 | 0.6 | Follower |
| 6 | ll | 0.1666 | 0.6789 | 0.02 | Recruiter |
| 7 | l'f | 0.1538 | 0.7654 | 0.6 | Recruiter |
| 8 | bl' | 0.1428 | 0.6543 | 0.6 | Recruiter |
| 9 | b'r' | 0.3333 | 0.1234 | 0 | Recruiter |
| 10 | - | 0.2857 | 0.5432 | 0.6 | Follower |
| 11 | - | 0.2222 | 0.0234 | 0.6 | Follower |
| 12 | - | 0.1818 | 0.2345 | 0.6 | Follower |

Probabilitas *Pi* setiap lebah *recruiter* dihitung berdasarkan profitabilitasnya kemudian dimasukkan ke *Roulette Wheel*. Kemudian setiap lebah *follower* memilih jalur baru dari lebah-lebah *recruiter* dengan seleksi *Roulette Wheel* diperlihatkan pada Tabel III.21.

**Tabel III.21 Wheel follower ke recruiter tahap backward pass ke-dua**

| Lebah | Simpul | *Pfi* | *Pi* | Roulette |
|---|---|---|---|---|
| 1 | u'f' | 0.1538 | 0.1159 | 0 - 0.1159 |
| 2 | dl | 0.2222 | 0.1675 | 0.1159 - 0.2834 |
| 4 | f'f' | 0.1538 | 0.1159 | 0.2834 - 0.3994 |
| 6 | ll | 0.1666 | 0.1256 | 0.3994 - 0.5250 |
| 7 | l'f | 0.1538 | 0.1159 | 0.5250 - 0.6410 |
| 8 | bl' | 0.1428 | 0.1076 | 0.6410 - 0.7487 |
| 9 | b'r' | 0.3333 | 0.2512 | 0.7487 - 1 |
| total | | 1.3266 | 1 | |

Seleksi *Roulette Wheel* dilakukan dengan membangkitkan 5 bilangan acak untuk 5 lebah *follower*, kemudian setiap lebah *follower* diberikan simpul lebah *recruiter* sesuai dengan bilangan acaknya. Hasil seleksi *Roulette Wheel follower* ke *recruiter* dapat dilihat pada Tabel III.22.

**Tabel III.22 Hasil Roulette Wheel follower ke recruiter tahap backward pass ke-dua**

| **Lebah *Recruiter*** | **Bilangan acak** | **Lebah *Follower*** | **Simpul yang dipilih** |
|---|---|---|---|
| 3 | 0.4321 | 6 | ll |
| 5 | 0.0234 | 1 | u'f' |
| 10 | 0.3456 | 4 | f'f' |
| 11 | 0.6543 | 8 | bl' |
| 12 | 0.8765 | 9 | b'r' |

6. Tahap *Forward Pass* ke-3

Langkah-langkah pada tahap ini sama dengan langkah-langkah yang dilakukan pada tahap *forward pass* ke 2 yaitu melakukan evaluasi untuk simpul-simpul *recruiter* yang berbeda.

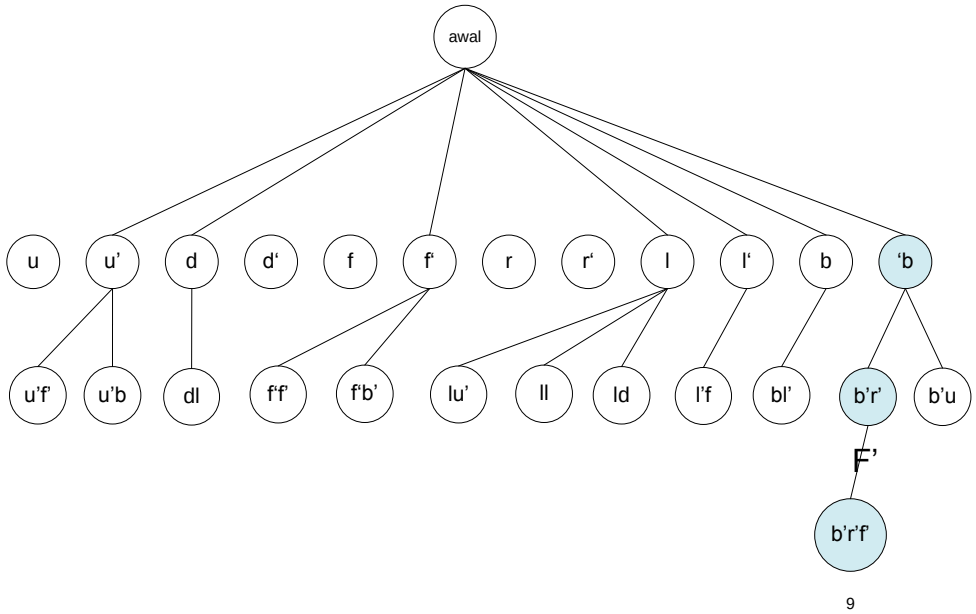
Evaluasi dilakukan secara terpisah sebanyak 7 kali untuk 7 simpul *recruiter* yang berbeda yaitu u'f', dl, f'f', ll, l'f, bl' dan b'r'.

Saat suksesor simpul b'r' untuk lebah 9 dan 12 dibangkitkan terdapat satu operator yang menuju ke simpul tujuan yaitu operator F' yang menuju ke simpul b'r'f' sehingga proses pencarian akan dihentikan. Tabel III.23 memperlihatkan proses evaluasi fungsi objektif suksesor simpul b'r'.

**Tabel III.23 Evaluasi fungsi objektif suksesor simpul b’r’**

| Simpul | Lebah | operator | *f* | *Pfi* | *Pi* |
|---|---|---|---|---|---|
| b'r' | 9 dan 12 | U | 4.5 | 0.222222 | 0.106101 |
| | | U' | 5 | 0.2 | 0.095491 |
| | | D | 4.5 | 0.222222 | 0.106101 |
| | | D’ | 5 | 0.2 | 0.095491 |
| | | F | 4 | 0.25 | 0.119363 |
| | | F’ | 0 | - | - |

Solusi kemudian direkonstruksi dengan merunut balik simpul *parent* dari simpul tujuan sampai ke simpul awal sehingga didapatkan solusinya adalah F’R’B’. Hasil akhir tahap ini dapat dilihat pada Gambar III.15.




**Gambar III.15 Lebah menemukan simpul tujuan**

Dari hasil analisis di atas dapat disimpulkan beberapa hal sebagai berikut:

1. Jumlah simpul yang dibangkitkan dan harus dievaluasi saat melakukan *forward pass* pada kasus terburuk, dengan asumsi semua lebah menjadi recruiter adalah 11 x jumlah lebah untuk setiap kedalaman. Bila jumlah kedalaman maksimum adalah 40 maka kebutuhan ruang memori untuk menampung informasi tentang simpul adalah 11 x 40 x jumlah lebah x ukuran simpul. Karena 1 konfigurasi Rubik's Cube membutuhkan 162byte memori, maka kebutuhan ruang memori minimal adalah 440 x 162 x jumlah lebah = jumlah lebah x 71280byte.

   Misalkan jumlah lebah adalah 5000 maka kebutuhan memori untuk menyimpan simpul yang ditelusuri adalah adalah 340 MB.
2. Jumlah simpul yang kemudian ditelusuri oleh lebah setelah *backward pass* tidak akan melebihi jumlah lebah yang ditentukan untuk setiap kedalaman.
3. Setelah tahap *forward pass* pertama, tahap *backward pass* dan *forward pass* kedua diiterasikan sampai ada lebah yang menemukan solusi, yaitu simpul yang berisi konfigurasi Rubik's Cube yang terselesaikan, atau bila nilai *counter* k lebih dari 40, yaitu dua kali 20 (*god's number*) karena operator 180° atau operator yang berlawanan pada 2 sisi yang berseberangan diabaikan.
4. Karena algoritma BCO bersifat probabilistik maka terdapat kemungkinan solusi tidak ditemukan dalam satu kali eksekusi, sehingga keseluruhan algoritma tersebut dapat diiterasikan sampai solusi ditemukan atau mencapai jumlah iterasi keseluruhan yang ditentukan sebelumnya atau dihentikan oleh user.

### III.2.6 Analisis Kebutuhan Non Fungsional

Analisis kebutuhan sistem non fungsional meliputi analisis masukan, analisis keluaran, perangkat keras (hardware), perangkat lunak (software), dan pengguna (user) sebagai bahan analisis kebutuhan yang harus dipenuhi dalam perancangan sistem yang akan diterapkan.

a. Analisis masukan

Analisis masukan dilakukan untuk mengetahui masukan data apa saja yang diperlukan oleh sistem. Masukan data yang diperlukan oleh sistem yaitu konfigurasi awal, konfigurasi tujuan, jumlah lebah dan jumlah iterasi.

b. Analisis keluaran

Analisis keluaran dilakukan untuk mengetahui keluaran dari sistem yang dibutuhkan oleh *user* yaitu solusi optimal berupa jalur terpendek serta pelacakan pencarian dari titik awal ke titik tujuan.

c. Analisis kebutuhan perangkat keras *(hardware)*

Agar aplikasi dapat berjalan dengan baik, maka dibutuhkan perangkat keras yang sesuai dengan kebutuhan aplikasi. Spesifikasi minimun perangkat keras yang dibutuhkan agar dapat menjalankan aplikasi ini secara optimal dapat dilihat pada Tabel III.24.

**Tabel III.24 Spesifikasi Minimum Perangkat Keras**

| Perangkat Keras | Spesifikasi |
|---|---|
| Prosesor | Prosesor 1.2 GHz |
| Resolusi Layar | 1366 x 768 *Pixel* |
| *Memory* | *Memory* 500MB |
| *Hard Disk* | *Hard Disk* 20MB |
| *Keyboard* dan *Mouse* | - |

d. Analisis kebutuhan perangkat lunak (*software*)

Kebutuhan perangkat lunak (*software*) untuk penerapan algoritma *BCO* dalam pencarian solusi optimal pada simulasi Rubik's Cube dapat dilihat pada Tabel III.25.

**Tabel III.25 Spesifikasi Perangkat Lunak**

| Perangkat Lunak | Spesifikasi |
|---|---|
| Sistem Operasi | Windows XP, Windows 7, Ubuntu 10.4. |
| Java Runtime | JRE 1.7.0_05 |

Pengguna diharapkan dapat memenuhi kebutuhan perangkat lunak yang telah direkomendasikan agar aplikasi dapat berjalan dengan baik.

e. Analisis pengguna

Pengguna yaitu peneliti yang dapat mengoperasikan komputer sehingga dapat menggunakan aplikasi yang akan dibangun. Pengguna juga diharapkan mengerti gerakan-gerakan dasar yang diperlukan untuk memainkan Rubik's Cube.

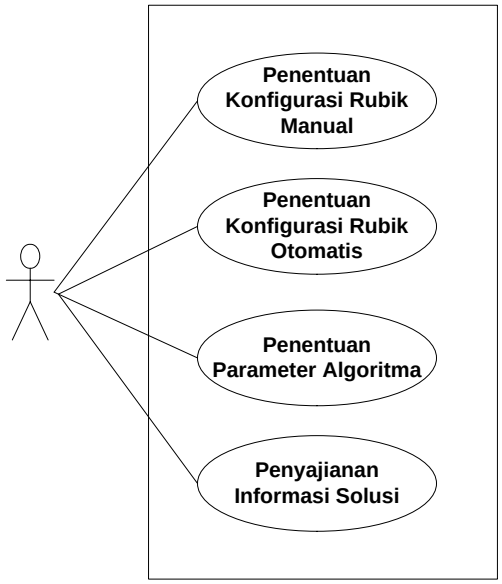
### III.2.7 Analisis Kebutuhan Fungsional

Pemodelan aplikasi yang akan dibangun dilakukan dengan metode analisis perancangan dan pengembangan perangkat lunak berorientasi objek dan menggunakan pemodelan *Unified Modeling Language* (*UML*).

Pemodelan mencakup *use case diagram, activity diagram, class diagram* dan *sequence diagram*. Proses yang dirancang diuraikan menjadi beberapa bagian yang dapat membentuk sistem tersebut menjadi satu kesatuan komponen.

### III.2.7.1 *Use Case Diagram* aplikasi Rubik's Cube dengan *BCO*

*Use case diagram* untuk memodelkan aplikasi untuk mencari langkah terpendek pada *puzzle* Rubik Cube's dengan algoritma BCO yang diusulkan dapat dilihat pada Gambar III.16.




**Gambar III.16 *Use Case Diagram* Aplikasi Penyelesaian Rubik's Cube dengan BCO**

### A. Definisi *Use Case*

Definisi *use case* berfungsi untuk menjelaskan proses yang terdapat pada setiap *use case.* Tabel III.26 menjelaskan definisi *Use Case* aplikasi Rubik's Cube dengan BCO.

**Tabel III.26 Definisi Use Case**

| ***Use Case*** | **Definisi** |
|---|---|
| Penentuan Konfigurasi Rubik Manual | Proses untuk menentukan konfigurasi Rubik's Cube yang akan dicari langkah solusinya secara manual |
| Penentuan Konfigurasi Rubik Otomatis | Proses untuk menentukan konfigurasi Rubik's Cube yang akan dicari langkah solusinya secara otomatis |
| Penentuan Parameter Algoritma | Proses untuk menentukan parameter dan menjalankan algoritma BCO |
| Penyajian Informasi Solusi | Proses untuk menampilkan hasil pencarian berupa solusi penyelesaian Rubik's Cube |

**B. Skenario (*Flow of Event*)**

Skenario merupakan penjelasan lebih detail dari *use case* dari awal sampai akhirnya diperoleh output.

1. Skenario Penentuan Konfigurasi Rubik Manual, diperlihatkan pada Tabel III.27.

**Tabel III.27 Skenario Penentuan Konfigurasi Rubik Manual**

| **Identifikasi** | | | |
|---|---|---|---|
| Nama Use Case | Penentuan Konfigurasi Rubik Manual | | |
| Tujuan | Memasukkan konfigurasi Rubik's Cube secara manual ke sistem | | |
| Aktor | User | | |
| **Skenario Utama** | | | |
| Kondisi Awal | Halaman Utama berisi Simulasi Rubik's Cube dalam kondisi terselesaikan, *combobox* mode konfigurasi tampil di layar | | |
| **No** | **Aksi Aktor** | **No** | **Respons Sistem** |
| 1 | Memilih mode konfigurasi Manual | 2 | Mengaktifkan mode konfigurasi manual |
| 1 | Menekan permukaan Rubik | 2 | Membaca koordinat mouse |
| | | 3 | Menyimpan koordinat mouse sebagai titik awal |
| | | 4 | Memetakan koordinat mouse ke permukaan rubik |
| 5 | Menggeser dan melepas mouse | 6 | Membaca koordinat mouse |
| | | 7 | Membaca titik awal |
| | | 8 | Membandingkan titik awal dengan koordinat terakhir |
| | | 9 | Mengubah konfigurasi rubik |
| | | 10 | Menampilkan konfigurasi rubik |
| Kondisi Akhir | Simulasi Rubik's Cube dengan konfigurasi yang dikehendaki user tampil di layar. | | |

2. Skenario Penentuan Konfigurasi Rubik Otomatis, diperlihatkan pada Tabel III.28

**Tabel III.28 Skenario Penentuan Konfigurasi Rubik Otomatis**

| **Identifikasi** | | | |
|---|---|---|---|
| Nama Use Case | Penentuan Konfigurasi Rubik Otomatis | | |
| Tujuan | Memasukkan konfigurasi Rubik's Cube secara otomatis ke sistem | | |
| Aktor | User | | |
| **Skenario Utama** | | | |
| Kondisi Awal | Halaman Utama berisi Simulasi Rubik's Cube dalam kondisi terselesaikan, *combobox* mode acak tampil di layar | | |
| **No** | **Aksi Aktor** | **No** | **Respons Sistem** |
| **Skenario Alternatif 1 (Memilih Acak Otomatis)** | | | |
| 1 | Memilih mode konfigurasi Otomatis | 2 | Mengaktifkan mode konfigurasi otomatis |
| 3 | Memasukkan Jumlah Langkah Acak | | |
| 4 | Menekan tombol Acak Rubik | 5 | Validasi masukan Jumlah Langkah Acak |
| | | 6 | Menjalankan prosedur acak |
| | | 7 | Mengubah konfigurasi rubik |
| | | 6 | Menampilkan konfigurasi rubik |
| Kondisi Akhir | Simulasi Rubik's Cube dengan konfigurasi acak tampil di layar. | | |

3. Skenario Penentuan Parameter Algoritma, diperlihatkan pada Tabel III.29.

**Tabel III.29 Skenario Penentuan Parameter Algoritma**

| **Identifikasi** | | | |
|---|---|---|---|
| Nama Use Case | Menentukan Parameter Algoritma | | |
| Tujuan | Menentukan parameter dan menjalankan algoritma BCO | | |
| Aktor | User | | |
| **Skenario Utama** | | | |
| Kondisi Awal | Panel parameter algoritma aktif | | |
| No | Aksi Aktor | No | Respons Sistem |
| 1 | Memasukkan Jumlah Lebah | | |
| 2 | Memasukkan Jumlah Iterasi | | |
| 3 | Menekan tombol Cari Solusi | 4 | Validasi masukan Jumlah Lebah dan Jumlah Iterasi |
| | | 5 | Membaca konfigurasi Rubik |
| | | 6 | Menjalankan algoritma BCO sesuai parameter |
| Kondisi Akhir | Log field berisi *progress* algoritma yang sedang atau telah berjalan tampil di layar | | |

4. Skenario Penyajian Informasi Solusi, diperlihatkan pada Tabel III.30.

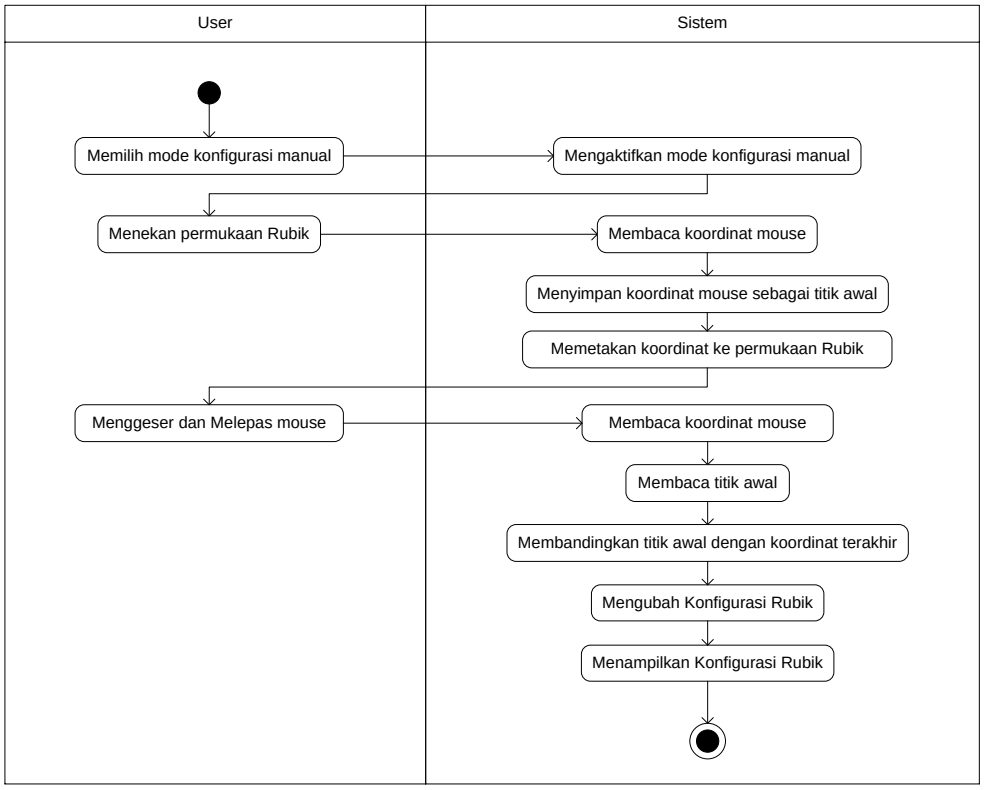
**Tabel III.30 Skenario Penyajian Informasi Solusi**

| **Identifikasi** | | | |
|---|---|---|---|
| Nama Use Case | Menampilkan Solusi | | |
| Tujuan | Menampilkan solusi yang dihasilkan oleh pencarian dengan algoritma BCO | | |
| Aktor | User | | |
| **Skenario Utama** | | | |
| Kondisi Awal | Halaman Utama tampil di layar | | |
| No | Aksi Aktor | No | Respons Sistem |
| 1 | Menekan tombol Berikutnya | 2 | Membaca konfigurasi solusi berikutnya |
| | | 3 | Menampilkan konfigurasi solusi berikutnya |
| 4 | Menekan tombol Sebelumnya | 5 | Membaca konfigurasi solusi sebelumnya |
| | | 6 | Menampilkan konfigurasi solusi sebelumnya |
| Kondisi Akhir | Konfigurasi Rubik’s Cube yang terselesaikan tampil di layar | | |

### III.2.7.2 *Activity Diagram*

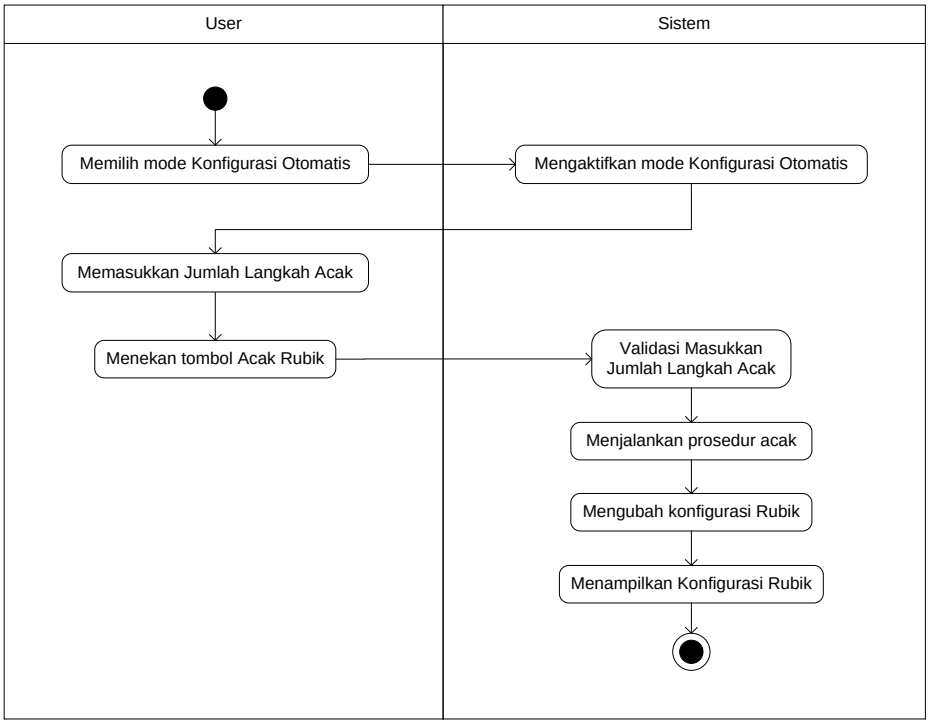
*Activity Diagram* menggambarkan berbagai alir aktivitas dalam sistem yang sedang dirancang, bagaimana masing-masing alir berawal, *decision* yang mungkin terjadi, serta proses paralel yang mungkin terjadi pada beberapa eksekusi.

1. *Activity Diagram* Penentuan Konfigurasi Rubik Manual, diperlihatkan pada Gambar III.17.



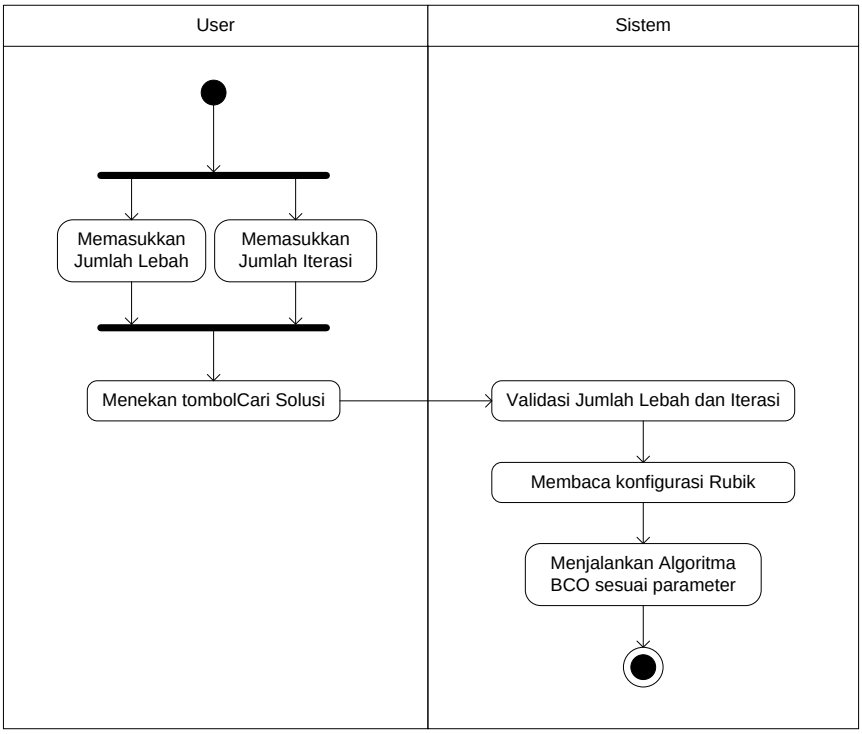

**Gambar III.17 *Activity Diagram* Penentuan Konfigurasi Rubik Manual**

2. *Activity Diagram* Penentuan Konfigurasi Rubik Otomatis, diperlihatkan pada Gambar III.18



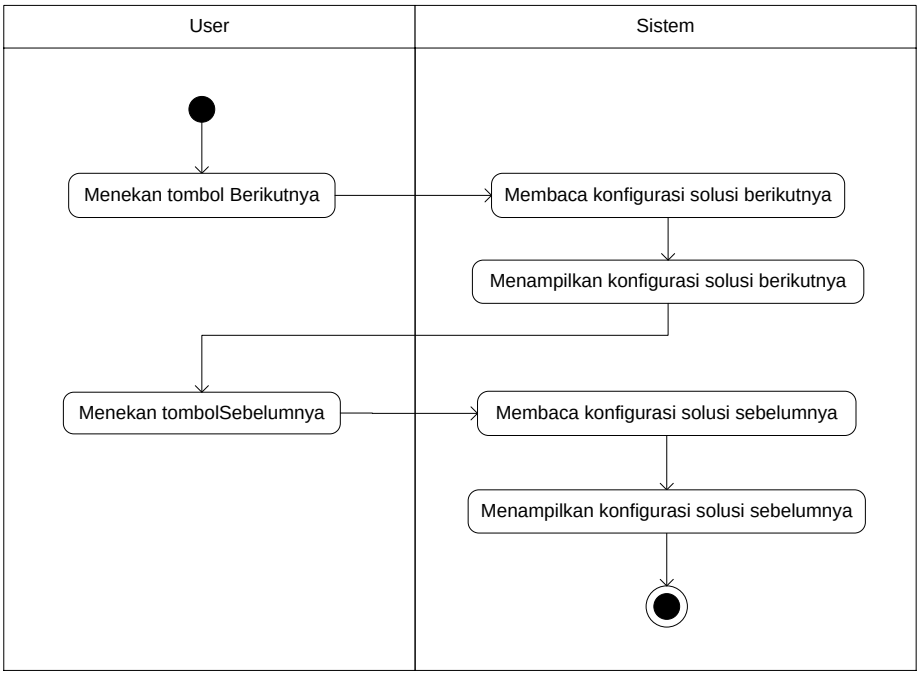

**Gambar III.18 *Activity Diagram* Penentuan Konfigurasi Rubik Otomatis**

3. *Activity Diagram* Penentuan Parameter Algoritma, diperlihatkan pada Gambar III.19.




**Gambar III.19 *Activity Diagram* Penentuan Parameter Algoritma**

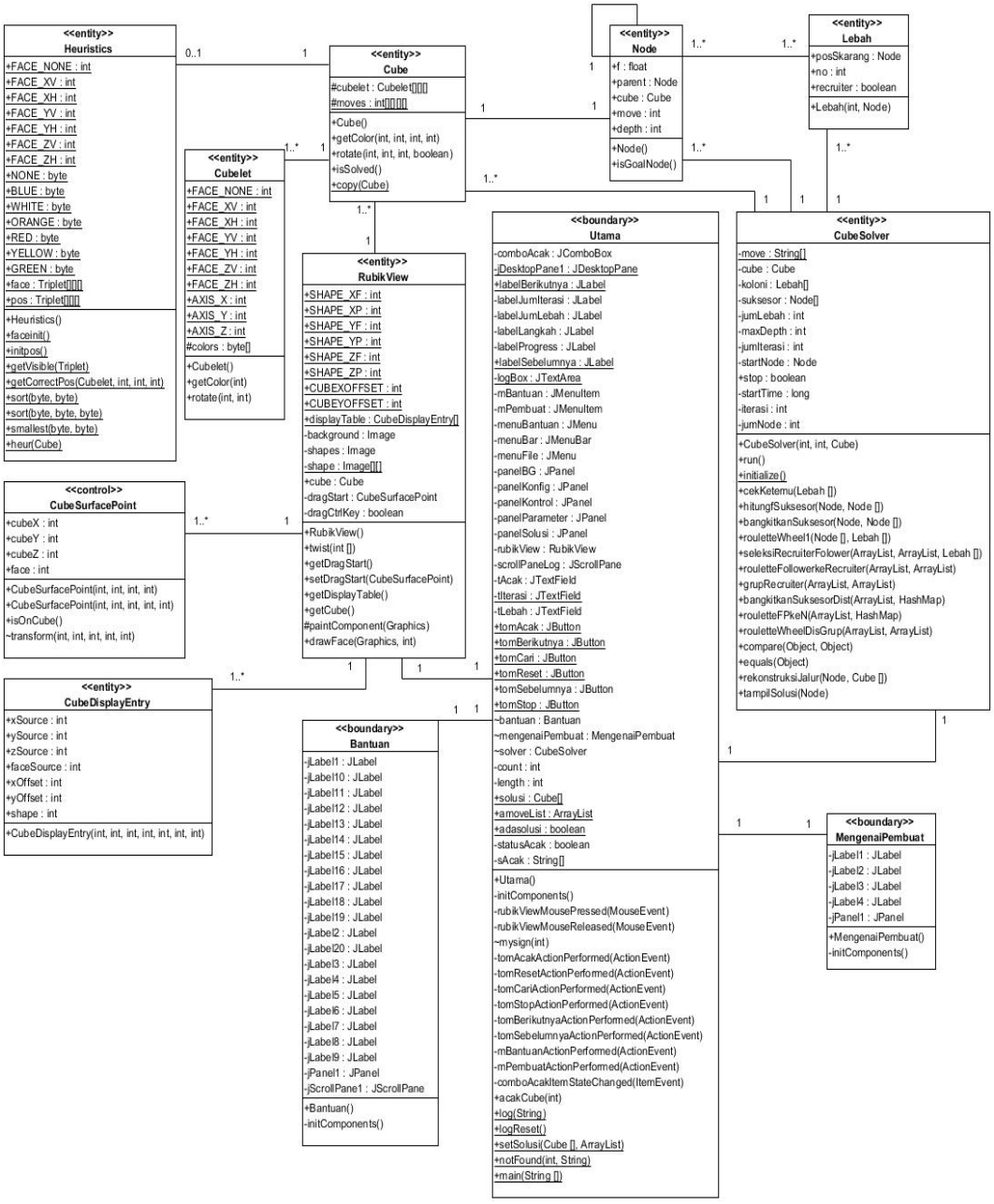
4. *Activity Diagram* Penyajian Informasi Solusi, diperlihatkan pada Gambar III.20.




**Gambar III.20 *Activity Diagram* Penyajian Informasi Solusi**

### III.2.7.3 *Class Diagram* Aplikasi Rubik's Cube dengan BCO

*Class Diagram* digunakan untuk menampilkan kelas-kelas dan paket-paket di dalam sistem, yang bertujuan untuk memberikan gambaran sistem secara statis serta relasi antar *class.* Class diagram aplikasi pencarian langkah terpendek pada Rubik's Cube dengan algoritma BCO diperlihatkan pada Gambar III.21.

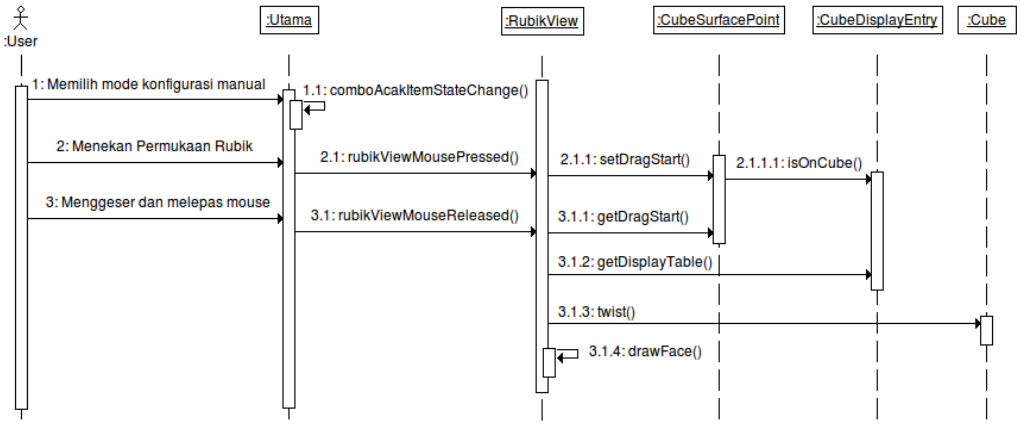



**Gambar III.21 *Class Diagram* Aplikasi Rubik's Cube dengan BCO**

### III.2.7.4 *Sequence Diagram*

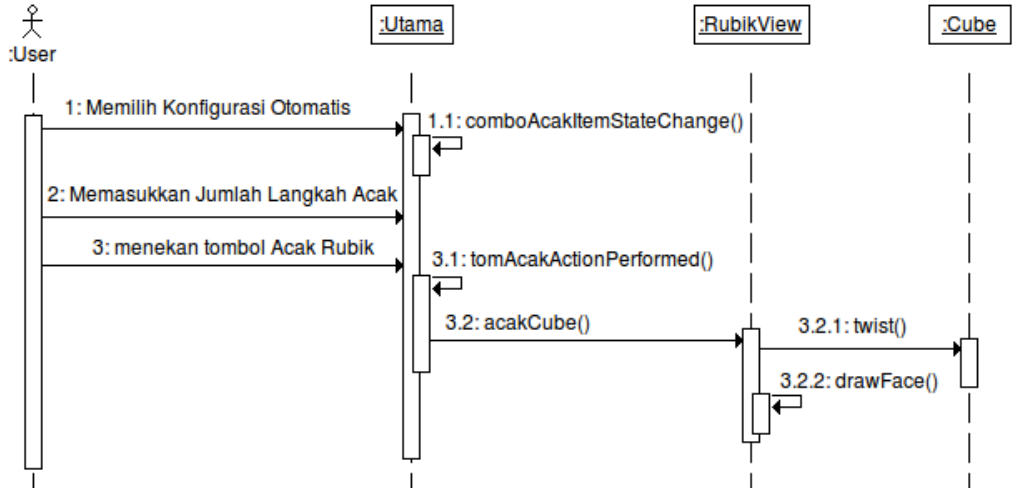
*Sequence Diagram* digunakan untuk menggambarkan interaksi antara sejumlah object dalam urutan waktu. Kegunaannya untuk menunjukkan rangkaian pesan yang dikirim antara object juga interaksi antar object yang terjadi pada titik tertentu dalam eksekusi sistem.

1. *Sequence Diagram* Penentuan Konfigurasi Rubik Manual, diperlihatkan pada Gambar III.22.



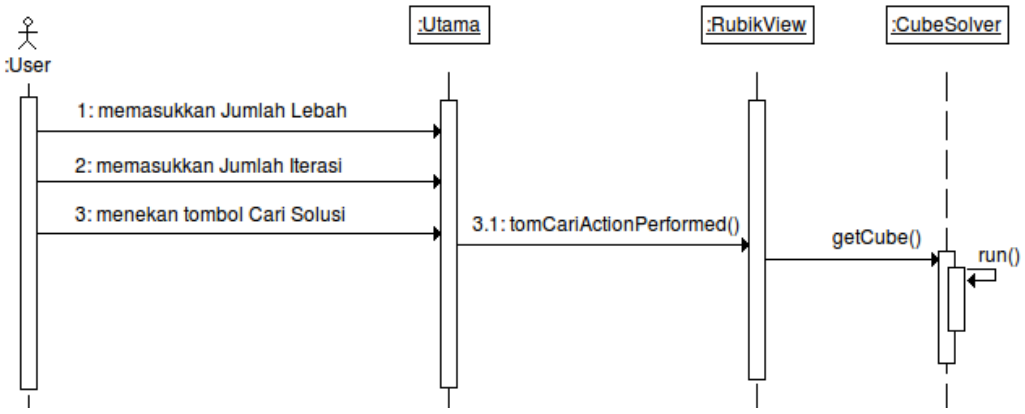

**Gambar III.22 *Sequence Diagram* Penentuan Konfigurasi Rubik Manual**

2. *Sequence Diagram* Penentuan Konfigurasi Rubik Otomatis, diperlihatkan pada Gambar III.23



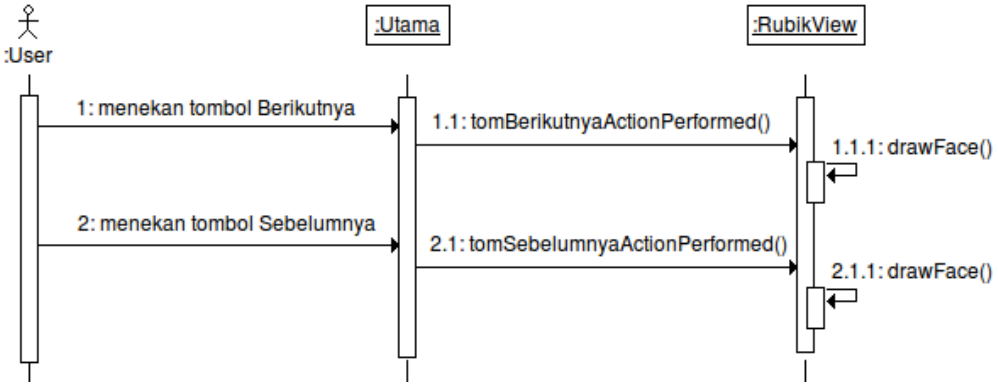

**Gambar III.23 Sequence Diagram Penentuan Konfigurasi Rubik Otomatis**

3. *Sequence Diagram* Penentuan Parameter Algoritma diperlihatkan pada Gambar III.24.




**Gambar III.24 *Sequence Diagram* Menentukan Parameter Algoritma**

4. *Sequence Diagram* Penyajian Informasi Solusi, diperlihatkan pada Gambar III.25.




**Gambar III.25 *Sequence Diagram* Penyajian Informasi Solusi**

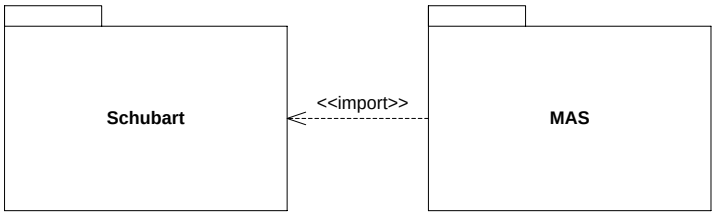
### III.2.7.5 *Package Diagram*

*Package diagram* digunakan untuk menggambarkan bagaimana elemen model diorganisasikan atau dikelompokkan ke dalam *packages.*

Terdapat 2 package yaitu package Schubart untuk menyimpan kelas-kelas dari applet simulasi Rubik's Cube oleh Michael Schubart yang digunakan untuk

memvisualisasikan Rubik's Cube dan package MAS untuk menyimpan kelas-kelas penerapan algoritma BCO.

Package Diagram untuk aplikasi yang akan dibangun dapat dilihat pada Gambar III.26.



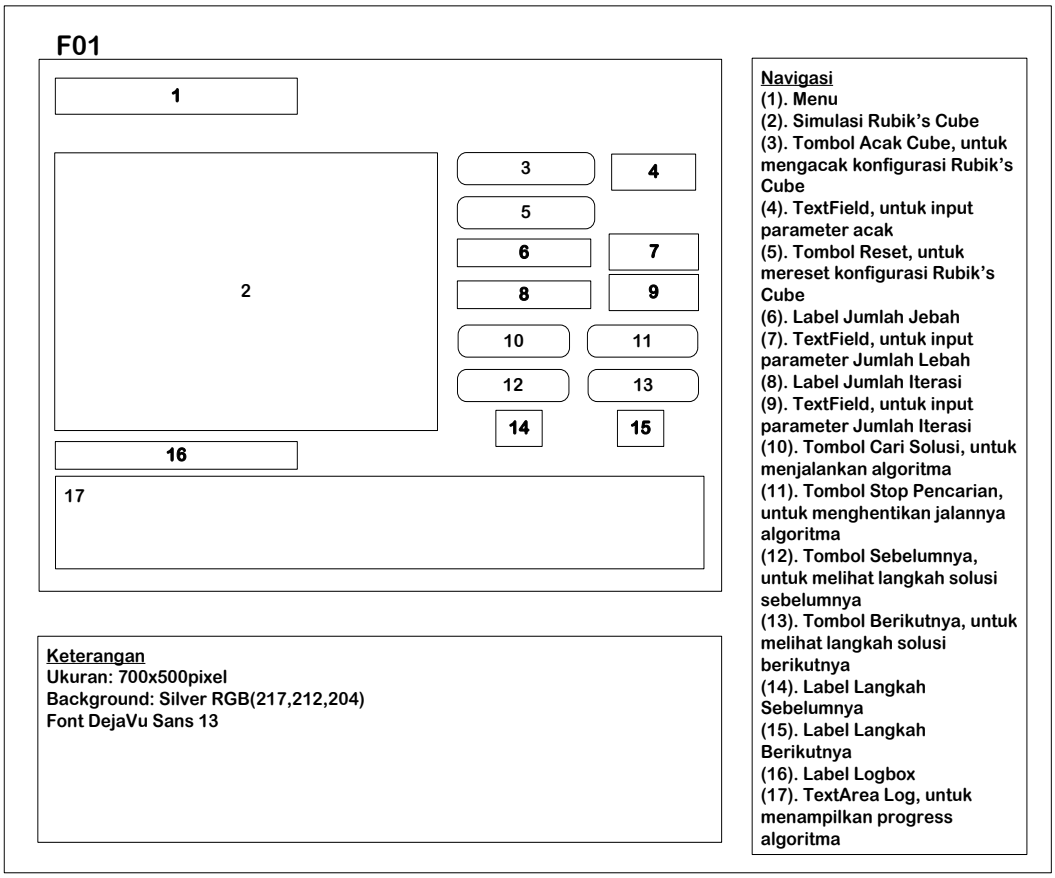

**Gambar III.26 Package Diagram**

## III.3 Perancangan Antarmuka

Perancangan antarmuka bertujuan untuk memberikan gambaran tentang aplikasi yang akan dibangun sehingga akan mempermudah saat mengimplementasikan dan membangun aplikasi. Perancangan antarmuka untuk aplikasi Rubik's Cube dengan BCO meliputi perancangan antarmuka *Form* Utama, *Form* Petunjuk dan *Form* Bantuan.

### III.3.1 Antarmuka *Form* Utama

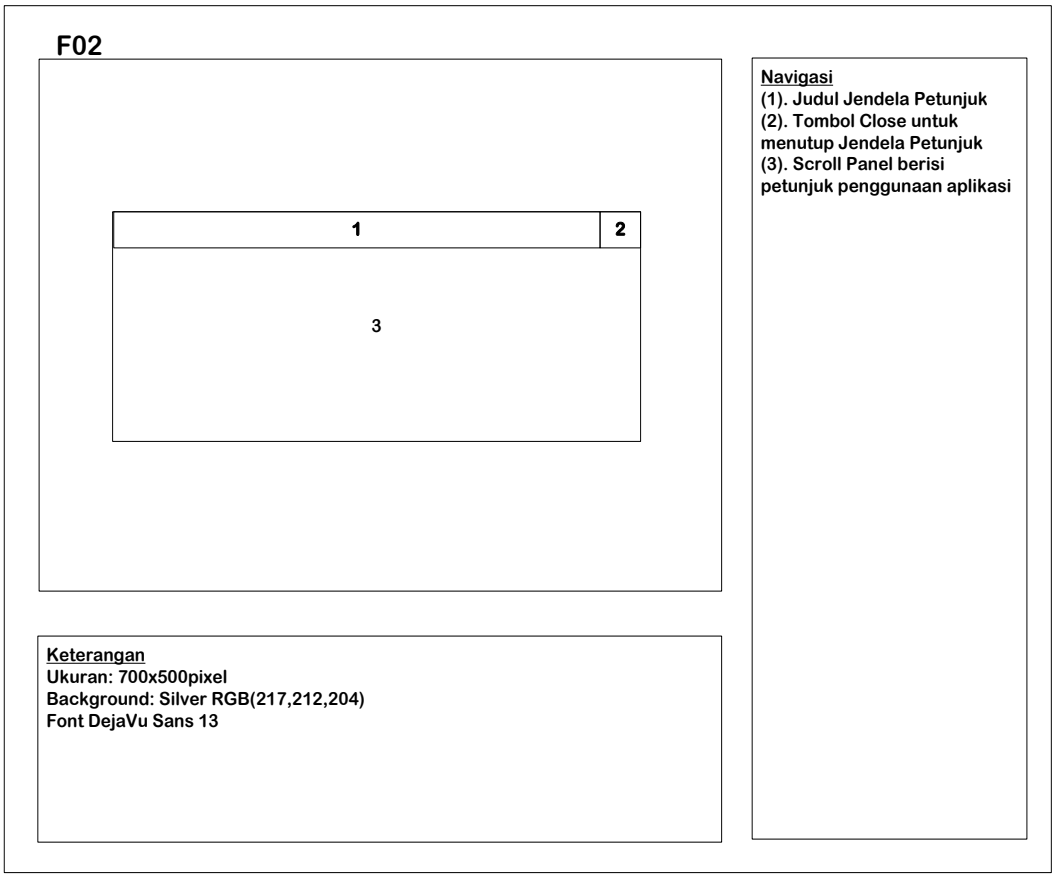
Perancangan antarmuka *Form* utama mendeskripsikan rencana tampilan *Form* dari awal aplikasi dijalankan. Gambaran perancangan antarmuka Jendela Utama dapat dilihat pada Gambar III.27.




**Gambar III.27 Perancangan antarmuka *Form* Utama**

### III.3.2 Antarmuka *Form* Petunjuk

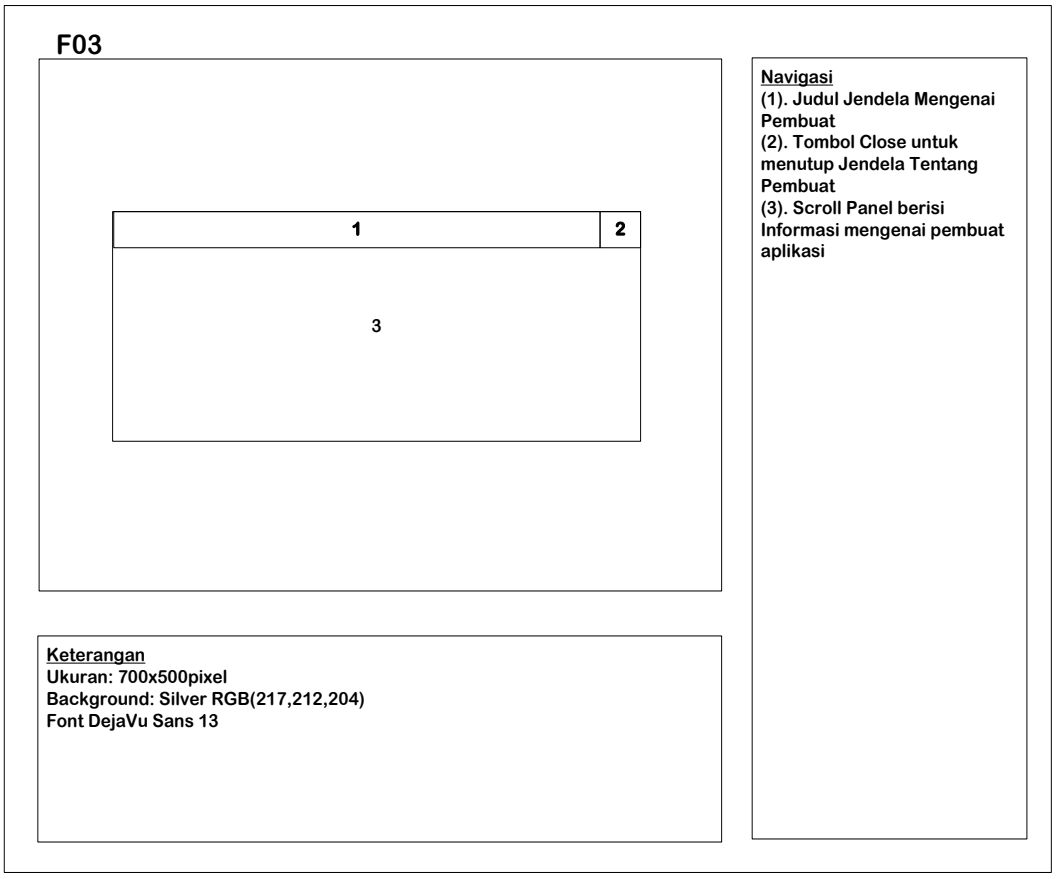
Perancangan antarmuka *Form* Petunjuk mendeskripsikan rencana tampilan dari petunjuk penggunaan aplikasi. Gambaran perancangan antarmuka *Form* Petunjuk dapat dilihat pada Gambar III.28.




**Gambar III.28 Perancangan antarmuka *Form* Petunjuk**

### III.3.3 Antarmuka *Form* Tentang Pembuat

Perancangan antarmuka *Form* Tentang Pembuat mendeskripsikan rencana tampilan informasi mengenai pembuat aplikasi. Gambar III.29 memperlihatkan gambaran dari perancangan antarmuka *Form* Tentang Pembuat.

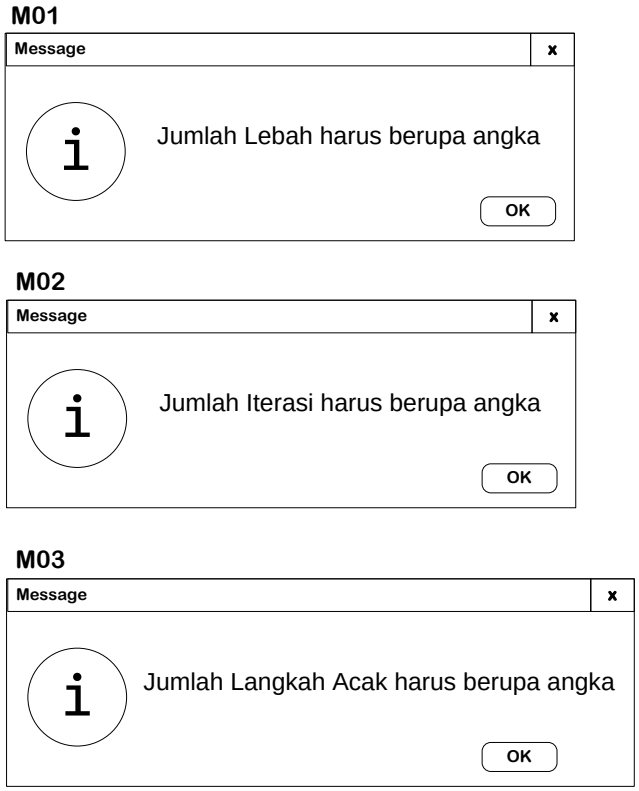



**Gambar III.29 Perancangan antarmuka *Form* Tentang Pembuat**

### III.3.4 Antarmuka Pesan Masukan Tidak Valid

Perancangan antarmuka pesan Tidak Valid mendeskripsikan rencana tampilan pesan bila parameter masukan Jumlah langkah acak, Jumlah lebah atau

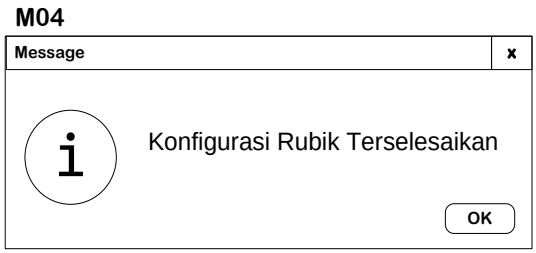
jumlah iterasi tidak valid. Gambar III.30 memperlihatkan gambaran dari perancangan antarmuka pesan Masukan Tidak Valid.




**Gambar III.30 Perancangan antarmuka Pesan Masukan Tidak Valid**

### III.3.5 Antarmuka Pesan Konfigurasi Rubik Terselesaikan

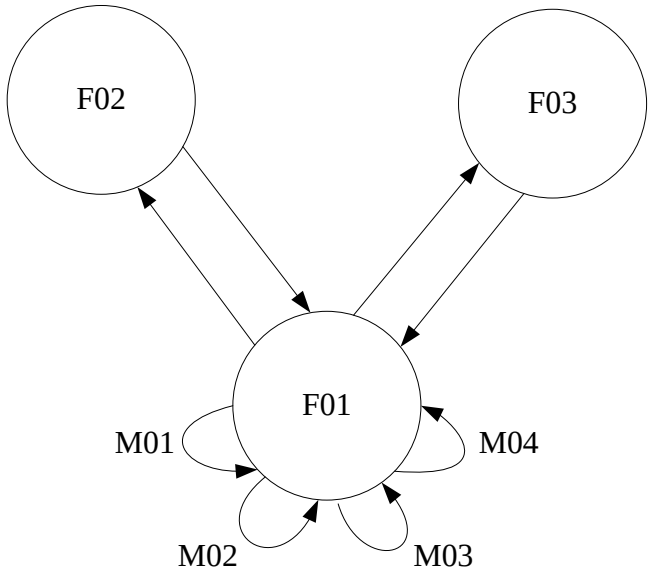
Perancangan antarmuka pesan Konfigurasi Rubik Terselesaikan mendeskripsikan rencana tampilan pesan bila konfigurasi Rubik telah dalam kondisi terselesaikan. Gambar III.31 memperlihatkan gambaran dari perancangan antarmuka pesan Konfigurasi Rubik Terselesaikan.




**Gambar III.31 Perancangan Antarmuka Pesan Konfigurasi Rubik Terselesaikan**

## III.4 Jaringan Semantik

Jaringan semantik menggambarkan alur informasi dari setiap antarmuka aplikasi. Jaringan semantik untuk aplikasi pencarian langkah solusi terpendek pada Rubik's Cube dapat dilihat pada Gambar III.32.



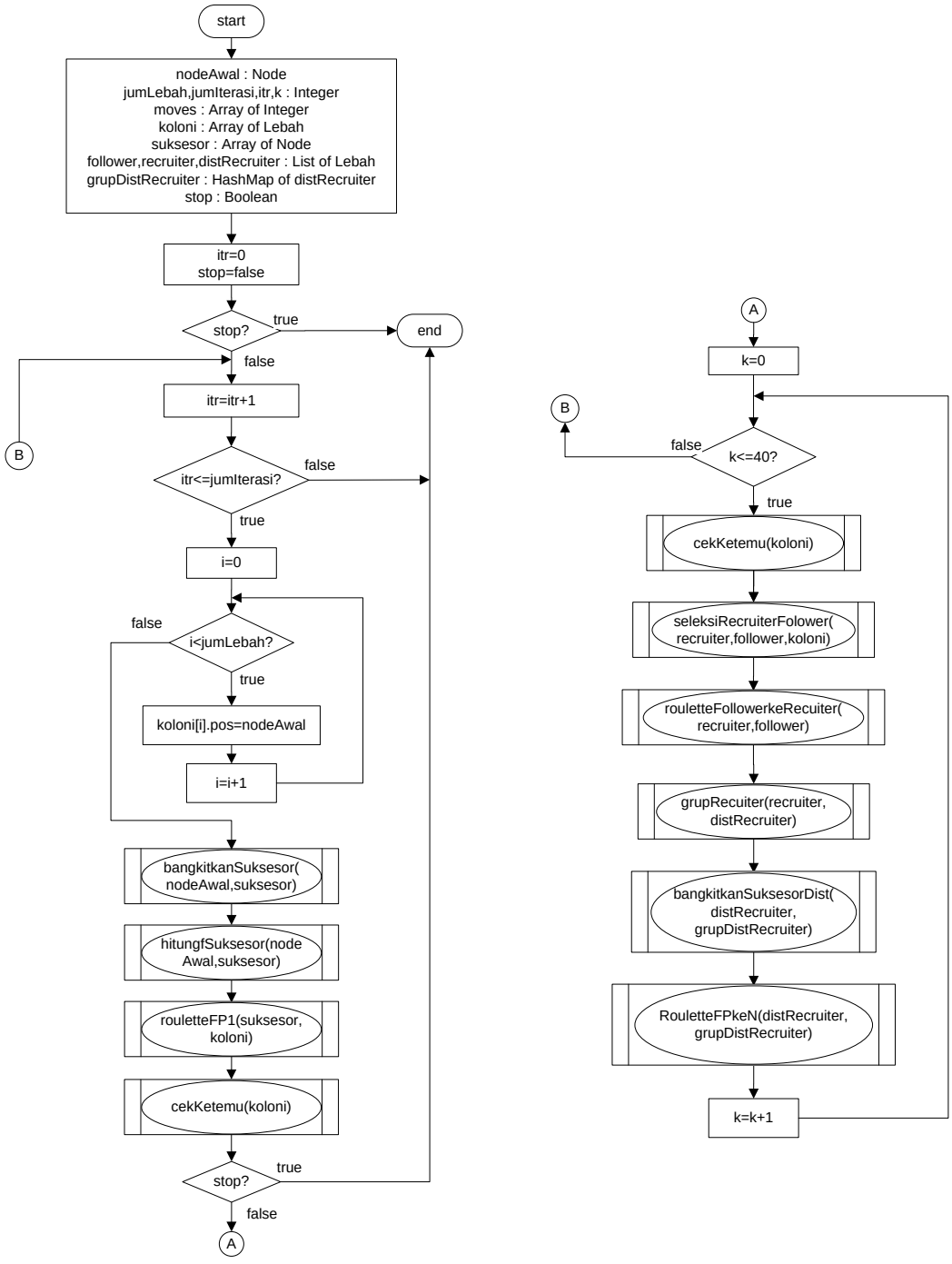

**Gambar III.32 Jaringan Semantik**

## III.5 Perancangan *Method*

Perancangan *method* merupakan perancangan yang dilakukan untuk menetapkan detail algoritma yang akan dinyatakan ke dalam suatu perangkat lunak. Perancangan *method* digambarkan dengan menggunakan *flowchart*

1. Perancangan *Method* run pada kelas CubeSolver.

   *Method* run mengimplementasikan *interface runnable* dan digunakan untuk menjalankan algoritma BCO dalam menyelesaikan konfigurasi Rubik's Cube yang telah ditentukan pada *thread* terpisah. *Method* ini langsung dijalankan saat kelas CubeSolver dinstansiasikan Untuk lebih jelasnya dapat dilihat pada Gambar III.33.

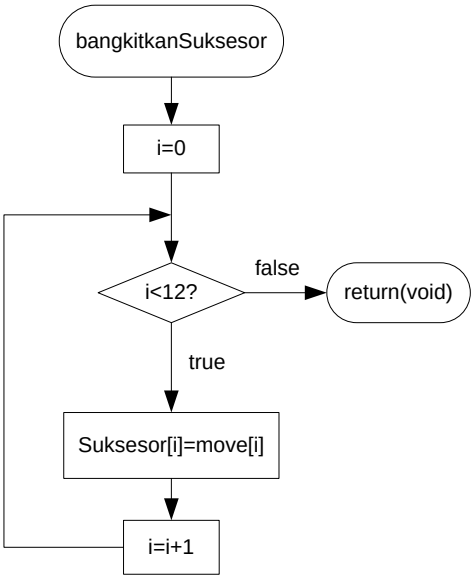



**Gambar III.33 Perancangan *Method* run di kelas CubeSolver**

Terdapat beberapa *method* yang dipanggil dalam *method* run yaitu :

a. *Method* bangkitkanSuksesor

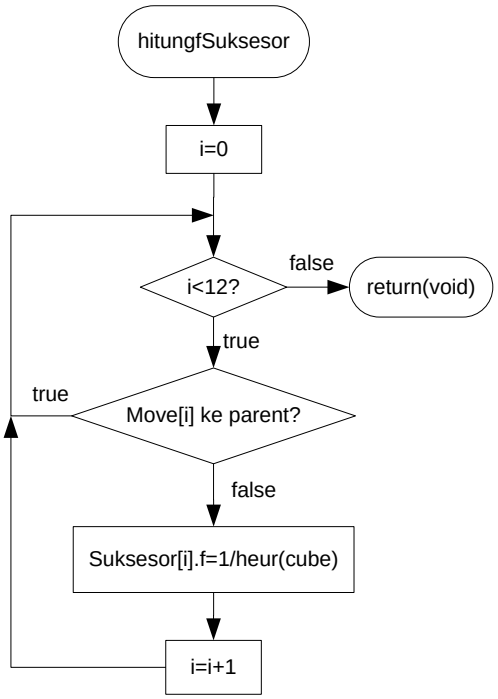
*Method* ini dijalankan untuk membangkitkan simpul-simpul suksesor dari suatu simpul, diperlihatkan pada Gambar III.34.




**Gambar III.34 *Flowchart Method* bangkitkanSuksesor**

b. *Method* hitungfSuksesor

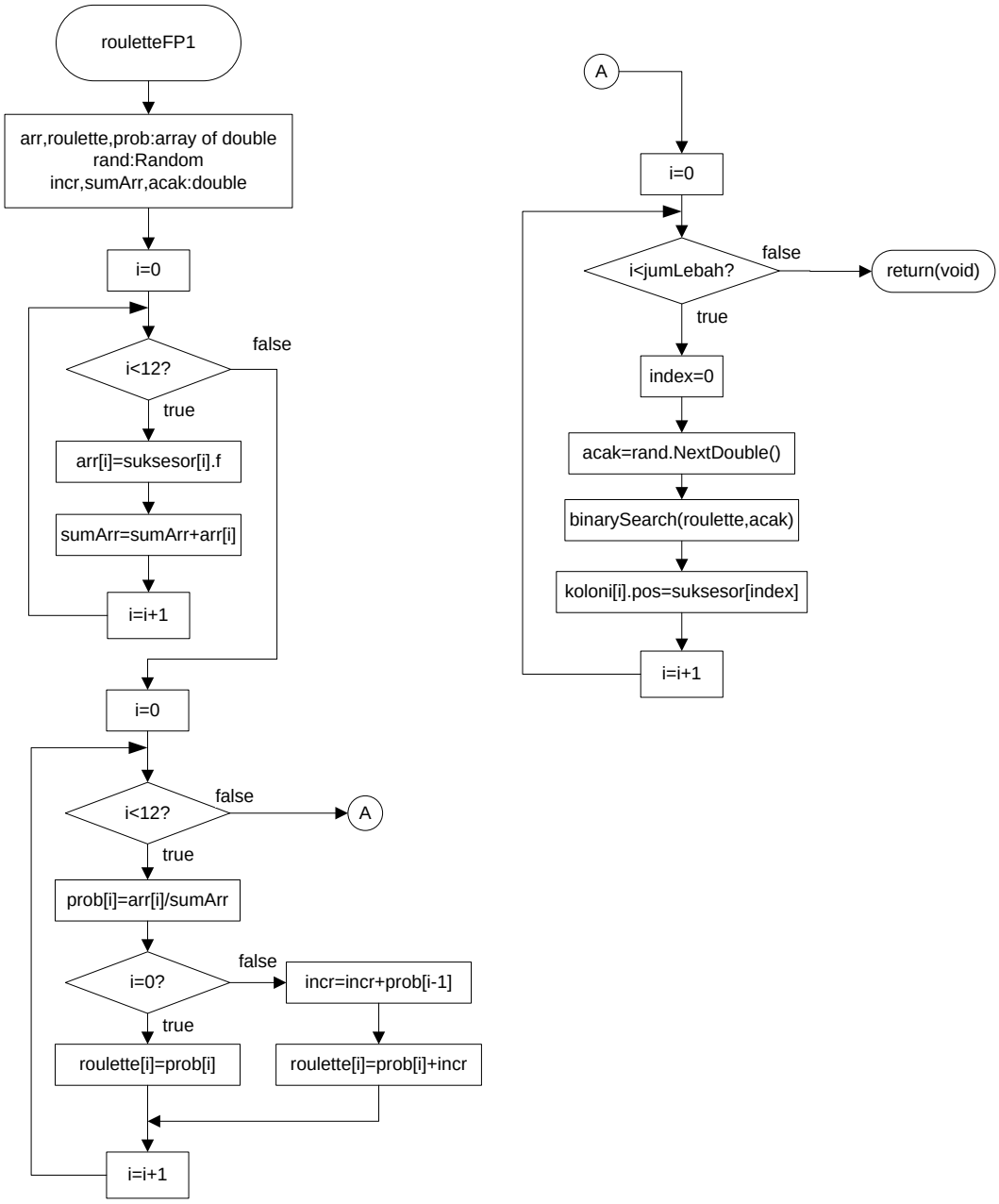
*Method* ini dijalankan untuk menerapkan batasan-batasan Rubik's Cube dan kemudian menghitung nilai profitabilitas setiap suksesor yang boleh ditelusuri, diperlihatkan pada Gambar III.35.




**Gambar III.35 *Flowchart Method* hitungfSuksesor**

c. *Method* rouletteFP1

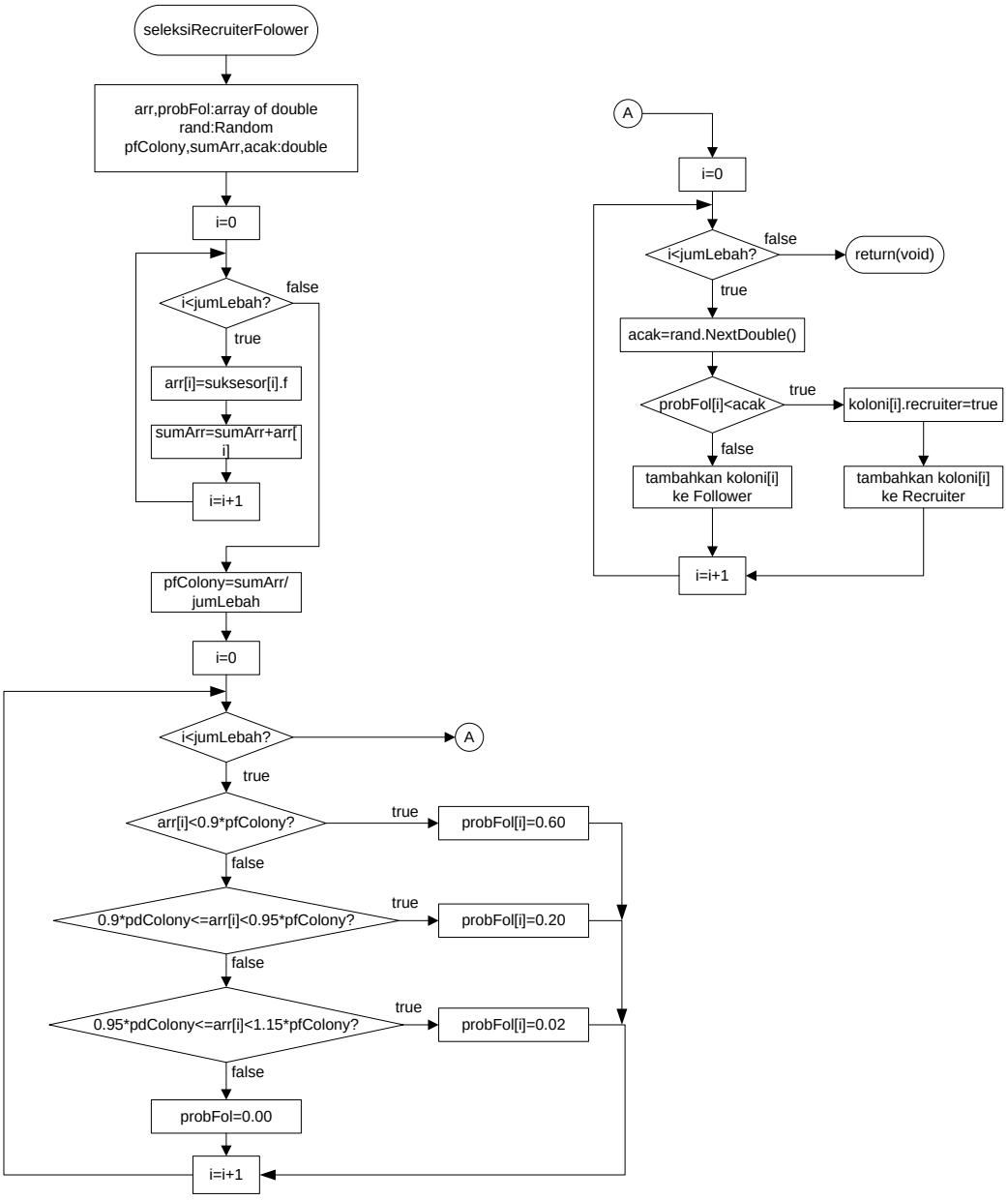
*Method* ini dijalankan untuk melakukan seleksi Roulette Wheel pada tahap *forward pass* pertama, diperlihatkan pada Gambar III.36.




**Gambar III.36 *Flowchart Method* rouletteFP1**

d. *Method* seleksiRecruiterFollower

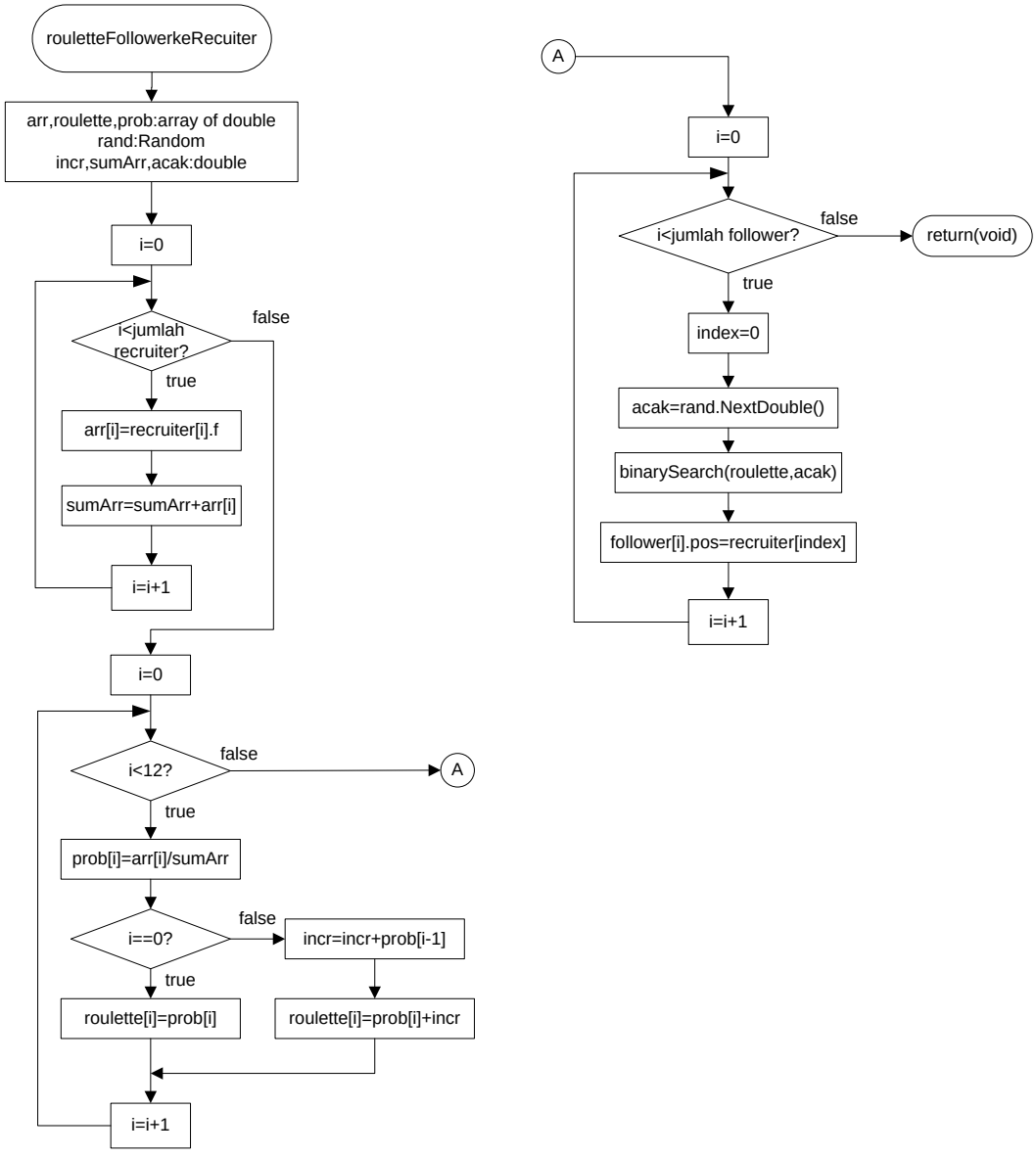
*Method* ini dijalankan untuk melakukan seleksi dimana lebah akan memilih apakah akan menjadi Follower atau Recruiter, diperlihatkan pada Gambar III.37.




**Gambar III.37 *Flowchart Method* seleksiRecruiterFollower**

e. *Method* rouletteFollowerkeRecruiter

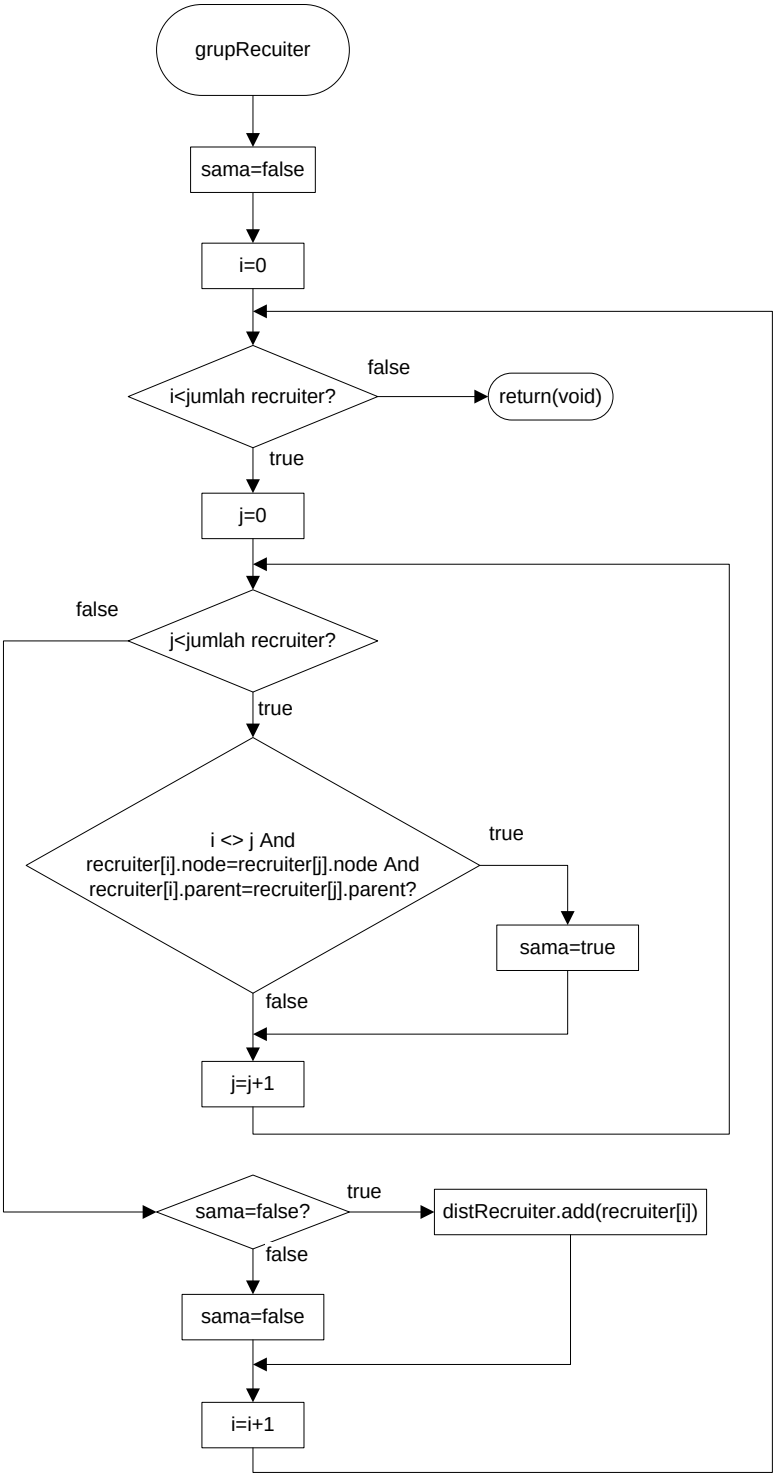
*Method* ini dijalankan untuk menentukan lebah Recruiter mana yang akan diikuti oleh setiap Follower melakukan seleksi Roulette Wheel oleh pada tahap *forward pass* pertama, diperlihatkan pada Gambar III.38.




**Gambar III.38 *Flowchart Method* rouletteFollowerkeRecruiter**

f. *Method* grupRecruiter

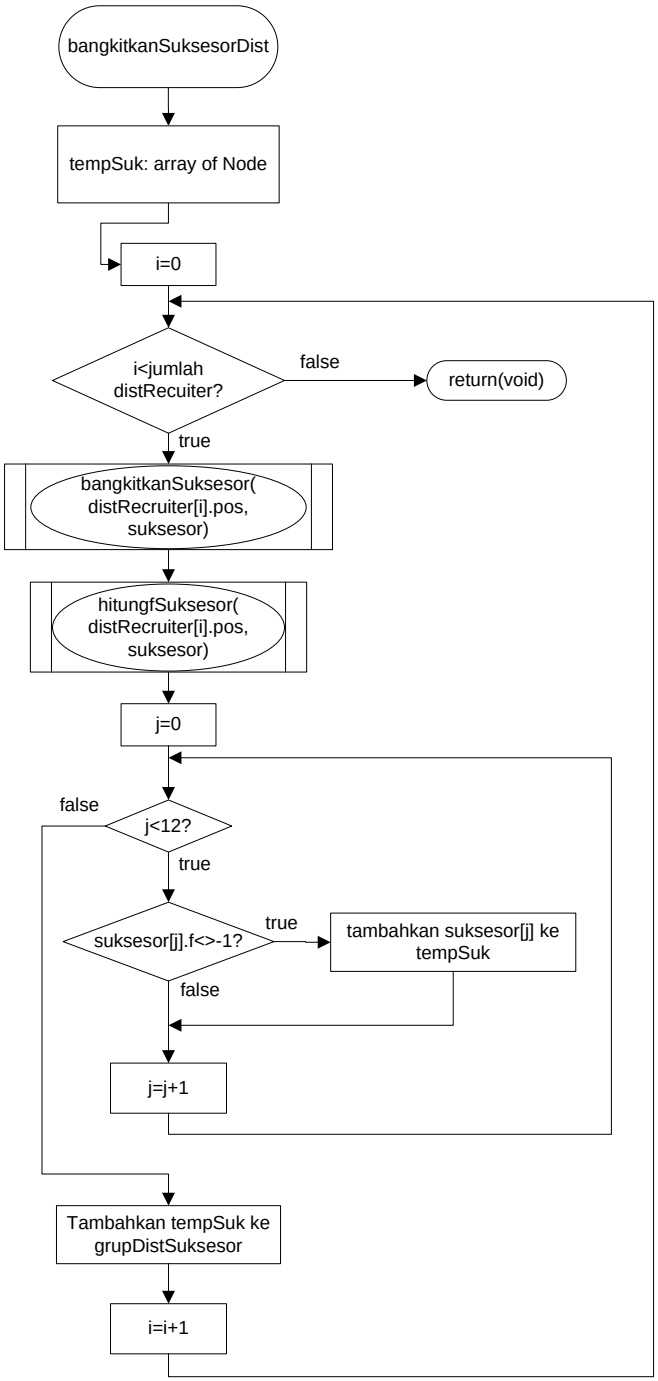
*Method* ini dijalankan untuk mengelompokkan lebah Recruiter sesuai simpulnya, diperlihatkan pada Gambar III.39.




**Gambar III.39 *Flowchart Method* grupRecruiter**

g. *Method* bangkitkanSuksesorDist

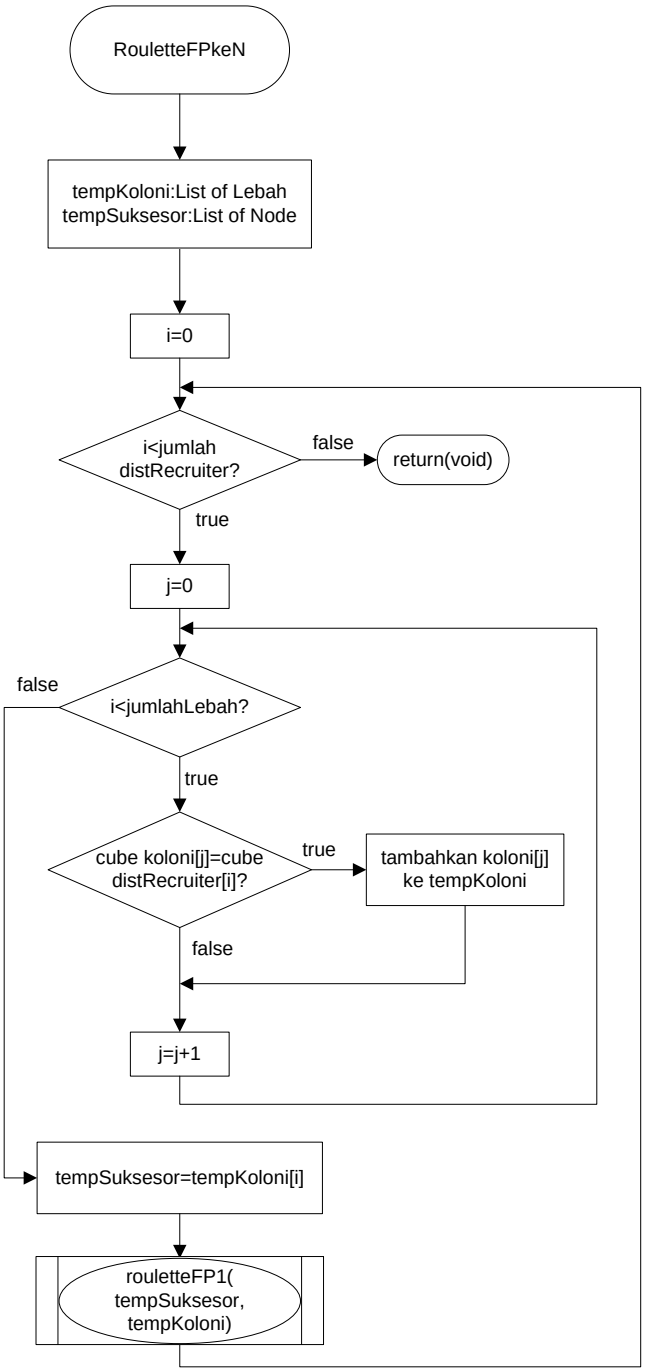
*Method* ini dijalankan untuk membangkitkan suksesor untuk setiap Recruiter dengan simpul yang berbeda, diperlihatkan pada Gambar III.40.




**Gambar III.40 *Flowchart Method* bangkitkanSuksesordist**

h. *Method* rouletteFPkeN

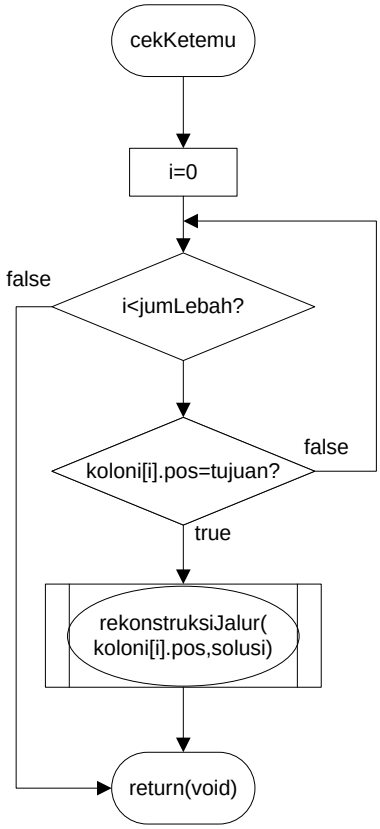
*Method* ini dijalankan untuk melakukan seleksi Roulette Wheel pada tahap *forward pass* kedua dan seterusnya, diperlihatkan pada Gambar III.41.




**Gambar III.41 *Flowchart Method* rouletteFPkeN**

i. *Method* cekKetemu

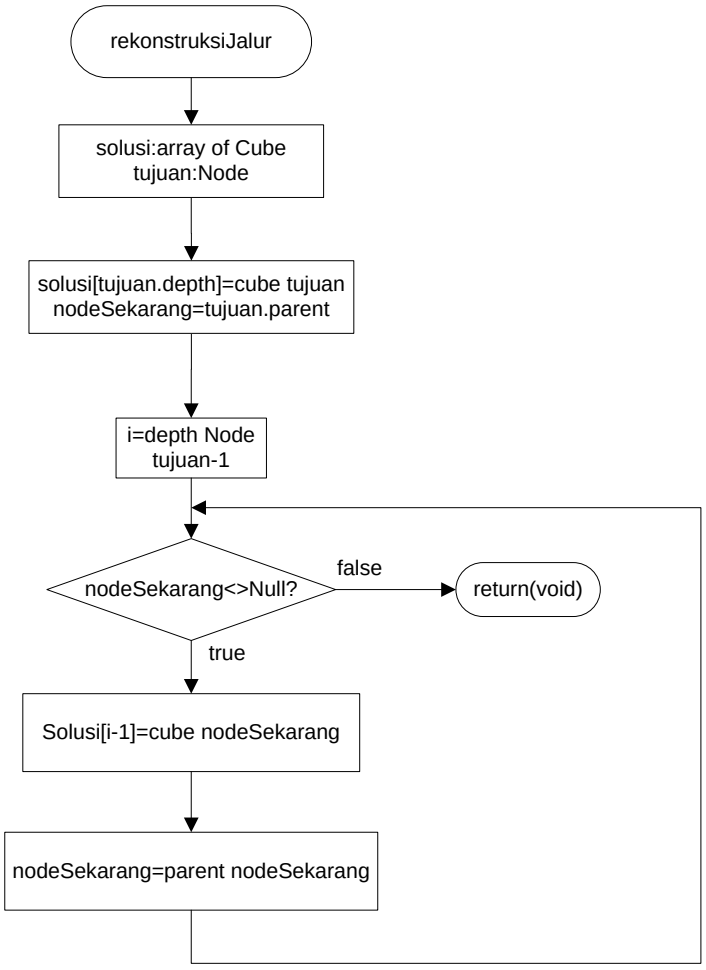
*Method* ini dijalankan untuk memeriksa apakah ada lebah yang memilih simpul tujuan, diperlihatkan pada Gambar III.42.




**Gambar III.42 *Flowchart Method* cekKetemu**

j. *Method* rekonstruksiJalur

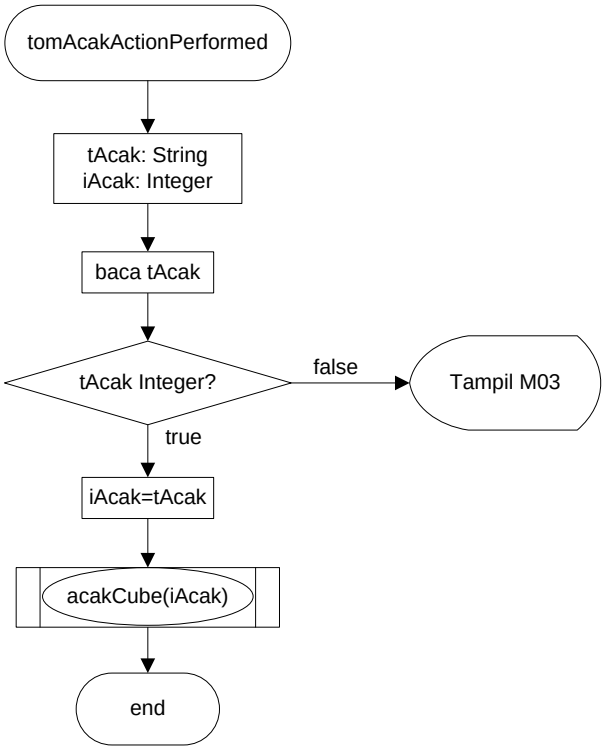
*Method* ini dijalankan untuk melakukan rekonstruksi jalur lebah yang pertama kali menemukan solusi dari simpul tujuan ke simpul awal, diperlihatkan pada Gambar III.43.




**Gambar III.43 *Flowchart Method* rekonstruksiJalur**

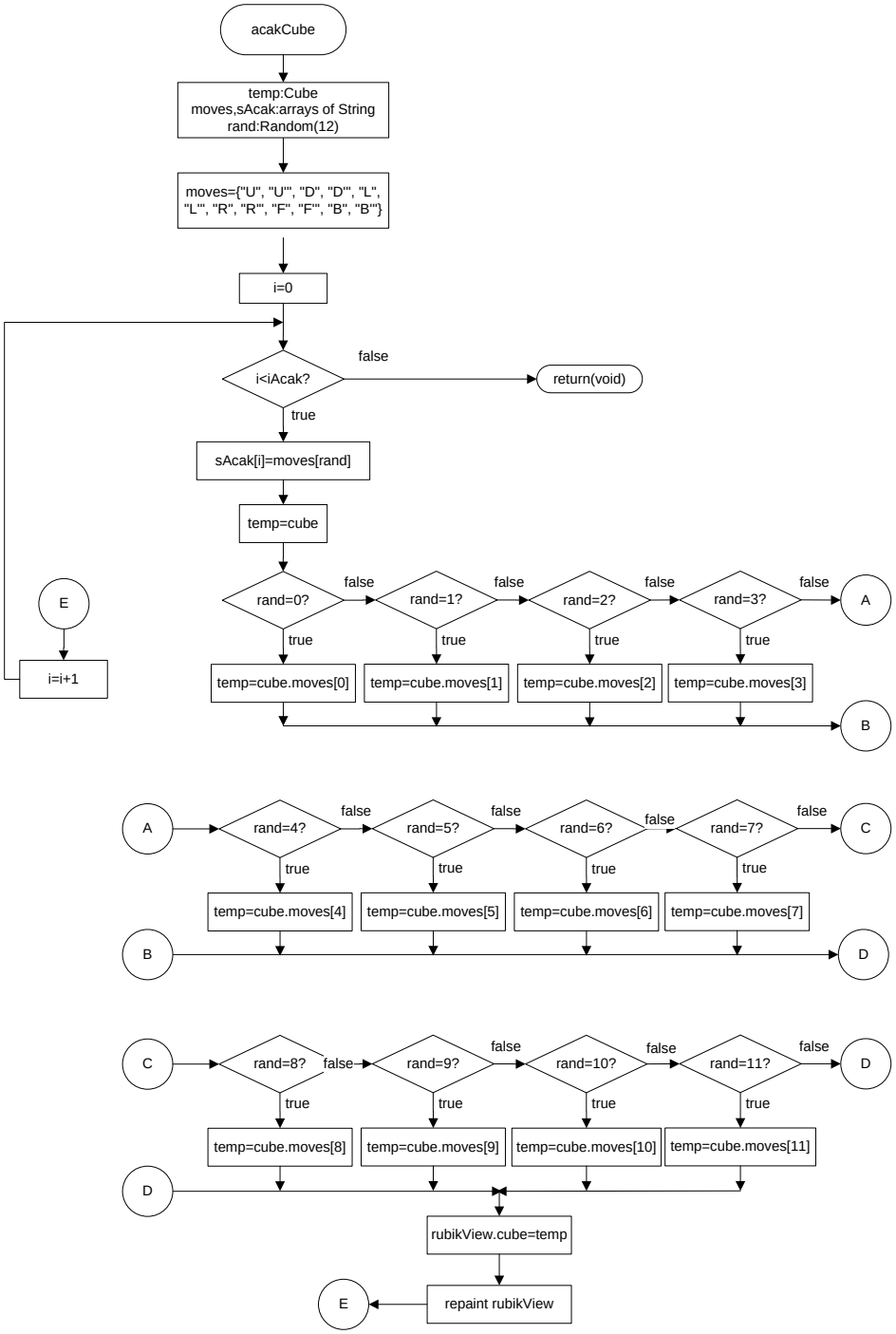
2. *Method* tomAcakActionPerformed pada kelas Utama

*Method* ini dijalankan untuk memvalidasi dan mengacak konfigurasi Rubik's Cube sesuai dengan langkah acak yang dikehendaki. Untuk lebih jelasnya dapat dilihat pada Gambar III.44




**Gambar III.44 *Flowchart Method* tomAcakActionPerformed**

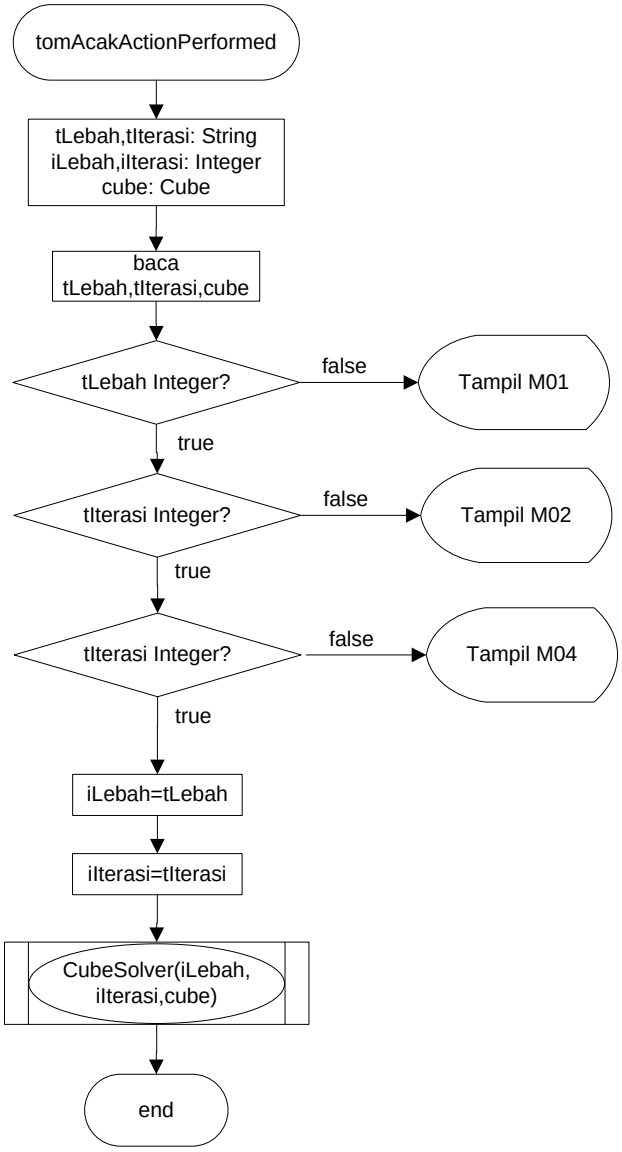
*Method* acakCube yang dipanggil dalam *method* tomAcakActionPerformed dapat dilihat pada Gambar III.45




**Gambar III.45 *Flowchart Method* acakCube**

3. *Method* tomCariActionPerformed pada kelas Utama

*Method* ini dijalankan untuk memvalidasi masukkan parameter algoritma dan kemudian melakukan pencarian solusi, dapat dilihat pada Gambar III.46




**Gambar III.46 *Flowchart Method* tomCariActionPerformed**