

Penerapan Algoritma *Greedy* untuk Meminimalkan *Maximum Flow Conflict* pada Jaringan *Optical Circuit Switch* (OXC)

Niko Samuel Simanjuntak - 13524029

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung, Jalan Ganesha 10 Bandung

E-mail: nikosimanjuntak71@gmail.com , 13524029@std.stei.itb.ac.id

Abstrak—Perkembangan komputasi kecerdasan buatan skala besar menuntut sistem interkoneksi jaringan yang mampu menangani komunikasi antarkomponen dengan latensi rendah dan *bandwidth* tinggi. *Optical Circuit Switch* (OXC) menjadi salah satu pendekatan potensial karena memiliki keunggulan dalam efisiensi daya, latensi, dan kapasitas *bandwidth*. Namun, karakteristik koneksi OXC yang bersifat satu-ke-satu menyebabkan pemanfaatan *bandwidth* tidak selalu optimal, terutama ketika banyak aliran komunikasi harus melewati *link* yang sama. Makalah ini membahas penerapan algoritma *greedy* sebagai heuristik untuk menentukan konfigurasi dan skema *routing* pada jaringan OXC dengan tujuan meminimalkan nilai *maximum flow conflict*, yaitu jumlah maksimum aliran yang melewati suatu *link* dalam jaringan. Permasalahan yang dikaji terinspirasi dari *OXC Ports Arrangement Problem* pada *ICPC 2025 Online Winter Challenge powered by Huawei*, dengan memanfaatkan format masukan, keluaran, dan *test case* dari soal tersebut sebagai dasar pengujian. Strategi *greedy* yang digunakan memilih rute terbaik secara lokal untuk setiap *flow* berdasarkan estimasi beban *link* dan dampaknya terhadap konflik jaringan. Hasil pengujian diharapkan menunjukkan sejauh mana pendekatan ini mampu menghasilkan solusi yang cepat dan cukup baik, meskipun tidak menjamin optimalitas global.

Kata Kunci—*Optical Circuit Switch*, OXC, algoritma *greedy*, heuristik, *routing*, *maximum flow conflict*

I. PENDAHULUAN

Perkembangan kecerdasan buatan modern tidak hanya bergantung pada kemampuan komputasi setiap perangkat, tetapi juga pada kemampuan jaringan dalam menghubungkan perangkat-perangkat tersebut secara efisien. Pelatihan model kecerdasan buatan berskala besar membutuhkan banyak kartu komputasi yang saling bertukar data dalam jumlah besar. Apabila komunikasi antarkomponen tidak berjalan lancar, peningkatan jumlah perangkat tidak selalu menghasilkan peningkatan kinerja yang sebanding karena jaringan dapat menjadi hambatan utama dalam sistem.

Salah satu pendekatan yang digunakan untuk meningkatkan efisiensi komunikasi pada klaster komputasi adalah *Optical Circuit Switch* (OXC). Teknologi ini memungkinkan pembentukan koneksi optik secara langsung sehingga berpotensi memberikan latensi rendah, konsumsi daya yang

lebih efisien, dan kapasitas *bandwidth* yang tinggi. Namun, berbeda dengan sakelar elektrik yang lebih fleksibel, koneksi pada OXC bersifat satu-ke-satu. Akibatnya, pemilihan pasangan porta dan jalur komunikasi harus dilakukan secara hati-hati agar kapasitas jaringan dapat dimanfaatkan dengan baik.

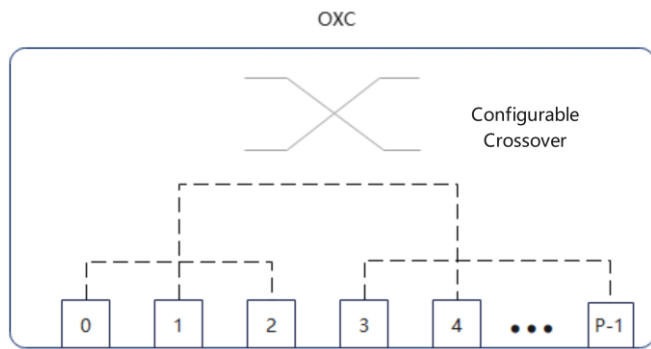
Tantangan utama muncul ketika banyak aliran komunikasi harus dilayani pada saat yang sama. Jika pemilihan rute tidak seimbang, beberapa tautan dapat menerima terlalu banyak aliran, sedangkan tautan lain kurang dimanfaatkan. Kondisi ini menimbulkan *maximum flow conflict*, yaitu nilai konflik terbesar pada suatu tautan dalam jaringan. Nilai ini penting karena menunjukkan seberapa parah penumpukan aliran pada bagian jaringan tertentu. Oleh karena itu, diperlukan strategi perutean yang mampu menyebarkan beban komunikasi secara lebih merata.

Makalah ini membahas penerapan algoritma *greedy* sebagai heuristik untuk meminimalkan *maximum flow conflict* pada jaringan *Optical Circuit Switch* (OXC). Pendekatan *greedy* dipilih karena sederhana, cepat, dan sesuai untuk persoalan yang memiliki banyak kemungkinan konfigurasi. Makalah ini terinspirasi dari *OXC Ports Arrangement Problem* pada *ICPC 2025 Online Winter Challenge powered by Huawei*. Melalui pendekatan ini, makalah bertujuan merancang strategi awal untuk menentukan konfigurasi dan skema perutean OXC secara efisien, meskipun tidak menjamin solusi optimal secara global.

II. DASAR TEORI

A. *Optical Circuit Switch* (OXC)

Optical Circuit Switch (OXC) adalah perangkat yang dapat membentuk koneksi optik antara satu porta internal dengan porta internal lainnya. Koneksi ini dibuat dengan mengatur cermin mikro untuk memantulkan cahaya, sehingga terbentuk jalur komunikasi dua arah antara dua porta.

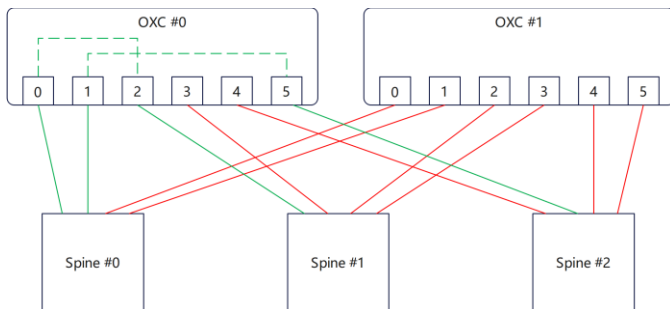


Gambar 1. Ilustrasi koneksi antarporta pada Optical Circuit Switch (OXC)
[Sumber: <https://codeforces.com/contest/2177/problem/A>]

Pada Gambar 1, OXC membentuk koneksi antara porta 0 dan porta 2, serta antara porta 1 dan porta 4. Artinya, perangkat yang terhubung ke porta 0 dapat berkomunikasi dengan perangkat pada porta 2, begitu pula perangkat pada porta 1 dengan perangkat pada porta 4. Hal penting dari OXC adalah setiap koneksi bersifat satu-ke-satu. Satu porta hanya dapat dipasangkan dengan satu porta lain pada satu waktu. Karakteristik inilah yang membuat konfigurasi OXC perlu ditentukan dengan tepat agar jalur komunikasi dapat dimanfaatkan secara efisien.

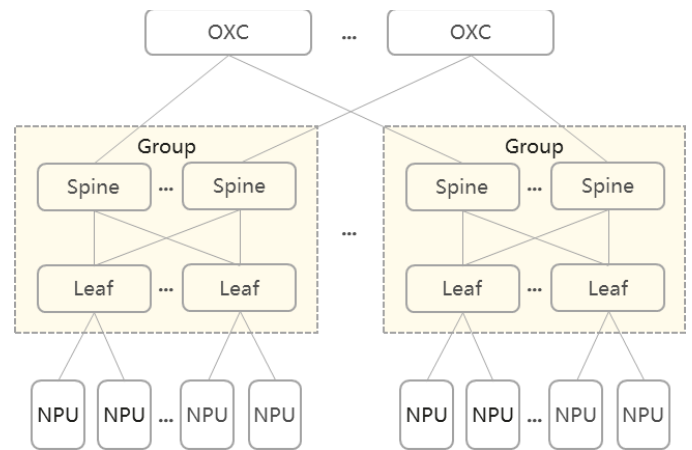
B. Arsitektur Jaringan OXC

Dalam arsitektur jaringan yang digunakan pada permasalahan ini, OXC tidak berdiri sendiri, tetapi terhubung dengan perangkat *Spine* yang berada di dalam beberapa *Group*. Setiap *Group* terdiri atas beberapa NPU, *Leaf*, dan *Spine*. Komunikasi antar-NPU dalam *Group* yang sama dapat diselesaikan melalui jaringan elektrik, sedangkan komunikasi antar-*Group* perlu melewati OXC.



Gambar 2. Contoh koneksi OXC dengan beberapa perangkat *Spine* [Sumber: <https://codeforces.com/contest/2177/problem/A>]

Pada Gambar 2, OXC #0 membentuk koneksi optik antara porta 0 dan porta 2, sehingga tercipta jalur komunikasi antara *Spine* #0 dan *Spine* #1. OXC yang sama juga menghubungkan porta 1 dan porta 5, sehingga *Spine* #0 dapat terhubung dengan *Spine* #2. Contoh ini menunjukkan bahwa koneksi antar-*Spine* dapat dibentuk dengan mengatur pasangan porta pada OXC.

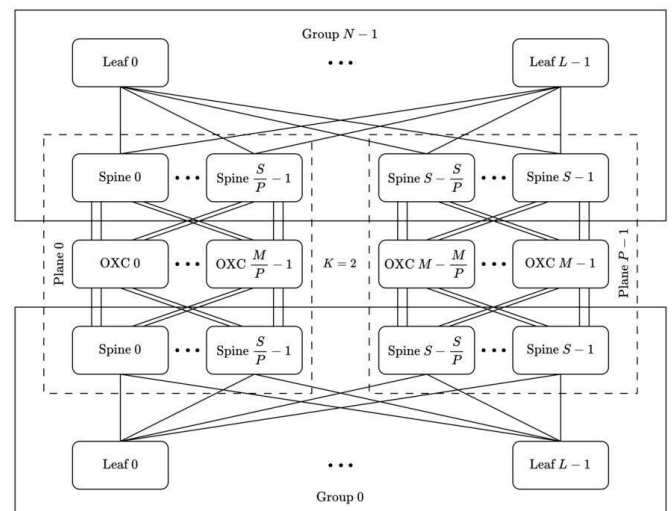


Gambar 3. Arsitektur jaringan dua tingkat elektrik dan satu tingkat optik
[Sumber: <https://codeforces.com/contest/2177/problem/A>]

Secara umum, jaringan terdiri atas dua tingkat sakelar elektrik, yaitu *Leaf* dan *Spine*, serta satu tingkat jaringan optik yang menggunakan OXC. Jalur komunikasi antar-*Group* melalui OXC dapat dituliskan sebagai berikut: NPU – *Leaf* – *Spine* – OXC – *Spine* – *Leaf* – NPU. Dengan demikian, OXC berperan sebagai lapisan inti yang menghubungkan komunikasi antar-*Group*, sedangkan komunikasi di dalam satu *Group* tetap ditangani oleh jaringan elektrik.

C. Jaringan Multi-Plane

Pada jaringan berskala besar, perangkat *Spine* dan OXC dapat dibagi ke dalam beberapa *plane* yang saling independen. Pembagian ini dilakukan agar jaringan dapat diperluas tanpa membuat seluruh perangkat harus saling terhubung secara langsung. Dengan adanya *plane*, setiap kelompok *Spine* hanya terhubung dengan OXC yang berada pada *plane* yang sama.



Gambar 4. Struktur jaringan multi-plane pada arsitektur OXC [Sumber: <https://codeforces.com/contest/2177/problem/A>]

Pada Gambar 4, jaringan dibagi menjadi beberapa *plane*. Di dalam setiap *plane*, setiap *Spine* terhubung dengan setiap OXC pada *plane* tersebut melalui sejumlah tautan. Sebaliknya, *Spine* dan OXC yang berada pada *plane* berbeda tidak memiliki tautan langsung. Hal ini membuat pemilihan rute tidak dapat dilakukan

secara sembarang karena suatu aliran hanya dapat melewati kombinasi *Spine* dan *OXC* yang berada dalam *plane* yang sesuai.

D. Model Komponen dan Penomoran Jaringan

Jaringan *OXC* pada permasalahan ini dimodelkan menggunakan beberapa komponen utama, yaitu *Group*, *Leaf*, *Spine*, *OXC*, *plane*, dan tautan. Terdapat (N) *Group* yang dinomori dari 0 sampai ($N-1$). Setiap *Group* memiliki (S) *Spine* dan (L) *Leaf*, sedangkan jumlah *OXC* dalam jaringan adalah (M).

Perangkat *Spine* dan *OXC* dibagi ke dalam (P) *plane*. Dalam satu *plane*, setiap *Spine* terhubung dengan setiap *OXC* pada *plane* yang sama melalui (K) tautan. Sebaliknya, *Spine* dan *OXC* pada *plane* berbeda tidak memiliki tautan langsung. Oleh karena itu, pemilihan rute harus memperhatikan kesesuaian *plane* antara *Spine* dan *OXC*.

Setiap *OXC* memiliki sejumlah porta yang dinyatakan dengan rumus:

$$R = N \cdot (S/P) \cdot K$$

Nilai (R) menunjukkan banyaknya porta pada satu *OXC*. Porta tersebut dipetakan berdasarkan nomor *Group*, nomor *Spine* dalam *plane*, dan nomor tautan. Model penomoran ini digunakan sebagai dasar untuk menentukan rute aliran komunikasi, yaitu memilih *Spine* asal, nomor tautan, *OXC*, *Spine* tujuan, dan nomor tautan tujuan.

E. Permintaan Aliran Komunikasi

Setiap tugas kecerdasan buatan dapat memiliki kebutuhan komunikasi yang berbeda antarkartu komputasi. Dalam permasalahan ini, kebutuhan tersebut dimodelkan sebagai sejumlah aliran komunikasi dua arah antar-*Leaf*. Setiap aliran menghubungkan dua *Leaf* yang berada pada *Group* berbeda sehingga komunikasi tidak dapat diselesaikan hanya melalui jaringan elektrik di dalam satu *Group*.

Sebuah aliran komunikasi dikirim melalui jalur:

$$LeafA - SpineA - OXC - SpineB - LeafB$$

Dengan demikian, untuk setiap aliran, perlu ditentukan *Spine* asal, *OXC* yang digunakan, dan *Spine* tujuan. Pemilihan jalur inilah yang menjadi bagian penting dalam proses perutean, karena beberapa aliran dapat menggunakan tautan yang sama dan menyebabkan penumpukan beban.

F. Maximum Flow Conflict

Maximum flow conflict adalah jumlah maksimum aliran yang melewati suatu tautan dalam jaringan. Nilai ini digunakan untuk mengukur seberapa besar penumpukan aliran pada tautan tertentu. Jika terlalu banyak aliran melewati tautan yang sama, nilai konflik menjadi tinggi dan pemanfaatan jaringan menjadi kurang seimbang.

Tujuan utama dari permasalahan ini adalah meminimalkan nilai *maximum flow conflict*. Dengan nilai konflik yang lebih kecil, beban komunikasi dapat tersebar lebih merata pada jaringan. Oleh karena itu, algoritma perutean tidak cukup hanya

mencari jalur yang valid, tetapi juga perlu mempertimbangkan beban tautan yang sudah digunakan oleh aliran lain.

G. Biaya Penyesuaian Topologi OXC

OXC dapat mengubah konfigurasi koneksi antarporta sesuai kebutuhan. Namun, perubahan konfigurasi tersebut memiliki biaya. Biaya penyesuaian topologi *OXC* dihitung berdasarkan perubahan dari topologi sebelumnya ke topologi baru. Tindakan seperti membentuk koneksi baru antarporta atau menghapus koneksi lama masing-masing memiliki biaya.

Dalam konteks makalah ini, biaya penyesuaian topologi menjadi pertimbangan tambahan. Fokus utama tetap berada pada upaya meminimalkan *maximum flow conflict*, tetapi konfigurasi yang terlalu sering berubah juga dapat menambah beban penyesuaian. Oleh karena itu, solusi yang baik sebaiknya tidak hanya menurunkan konflik aliran, tetapi juga menjaga agar perubahan topologi *OXC* tidak terlalu besar.

H. Algoritma Greedy sebagai Heuristik

Algoritma *greedy* adalah strategi penyelesaian masalah yang mengambil keputusan terbaik secara lokal pada setiap langkah. Dalam permasalahan ini, pendekatan *greedy* digunakan untuk memilih rute terbaik bagi setiap aliran berdasarkan kondisi jaringan saat itu. Rute yang dipilih adalah rute yang diperkirakan menghasilkan konflik paling kecil dibandingkan pilihan rute lainnya.

Pendekatan ini digunakan sebagai heuristik, yaitu metode yang tidak menjamin solusi optimal secara global, tetapi dapat menghasilkan solusi yang cukup baik dengan waktu komputasi yang lebih efisien. Hal ini sesuai dengan karakteristik permasalahan *OXC* yang memiliki banyak kemungkinan konfigurasi dan membutuhkan strategi perutean yang cepat. Dengan demikian, algoritma *greedy* dapat digunakan sebagai pendekatan awal untuk menentukan konfigurasi dan skema perutean pada jaringan *OXC*.

III. PEMODELAN MASALAH

A. Representasi Jaringan dan Aliran

Permasalahan pada makalah ini direpresentasikan sebagai jaringan yang terdiri atas beberapa komponen utama, yaitu *Group*, *Leaf*, *Spine*, *OXC*, *plane*, dan tautan. Setiap *Group* memiliki sejumlah *Leaf* dan *Spine*. Komunikasi di dalam *Group* yang sama dapat dilakukan melalui jaringan elektrik, sedangkan komunikasi antar-*Group* harus melewati *OXC* sebagai jaringan optik.

Setiap permintaan komunikasi dimodelkan sebagai sebuah *flow* dua arah antara dua *Leaf* yang berada pada *Group* berbeda. Sebuah *flow* dinyatakan dalam bentuk:

$$(g_A, l_A, g_B, l_B)$$

dengan g_A dan g_B menyatakan nomor *Group* asal dan tujuan, sedangkan l_A dan l_B menyatakan nomor *Leaf* asal dan tujuan. Karena kedua *Leaf* berada pada *Group* yang berbeda, maka aliran komunikasi harus melewati jalur *OXC*.

Secara umum, jalur yang dilalui oleh sebuah *flow* dapat dituliskan sebagai berikut:

$LeafA \rightarrow SpineA \rightarrow OXC \rightarrow SpineB \rightarrow LeafB$

Representasi ini menunjukkan bahwa setiap aliran tidak hanya membutuhkan pasangan *Leaf* asal dan tujuan, tetapi juga membutuhkan pemilihan *Spine* asal, OXC yang digunakan, serta *Spine* tujuan. Pemilihan komponen-komponen inilah yang nantinya menentukan beban pada setiap tautan dalam jaringan.

B. Keputusan Perutean

Setelah sebuah *flow* diketahui, langkah berikutnya adalah menentukan jalur yang akan digunakan oleh aliran tersebut. Dalam model ini, keputusan perutean untuk setiap *flow* dinyatakan dalam bentuk:

$$(x, k_x, m, y, k_y)$$

dengan x menyatakan *Spine* asal yang terhubung dengan *LeafA*, k_x menyatakan nomor tautan dari *Spine* asal menuju OXC, m menyatakan OXC yang digunakan, y menyatakan *Spine* tujuan yang terhubung dengan *LeafB*, dan k_y menyatakan nomor tautan dari OXC menuju *Spine* tujuan.

Keputusan tersebut harus memenuhi batasan jaringan. *Spine* yang dipilih harus berada pada *plane* yang sesuai dengan OXC yang digunakan. Selain itu, koneksi pada OXC harus mengikuti karakteristik satu-ke-satu. Dengan demikian, tidak semua kombinasi (x, k_x, m, y, k_y) dapat digunakan sebagai rute yang valid.

Keputusan perutean menjadi inti dari proses optimisasi. Pemilihan rute yang kurang tepat dapat menyebabkan banyak *flow* melewati tautan yang sama sehingga meningkatkan konflik aliran. Oleh karena itu, algoritma perlu memilih rute yang valid sekaligus mempertimbangkan beban tautan yang sudah digunakan oleh *flow* sebelumnya.

C. Fungsi Tujuan

Fungsi tujuan dalam pemodelan masalah ini adalah meminimalkan *maximum flow conflict*. Setiap kali sebuah *flow* melewati suatu tautan, beban pada tautan tersebut bertambah satu. Apabila banyak *flow* menggunakan tautan yang sama, maka beban tautan tersebut menjadi tinggi dan menimbulkan konflik aliran. Secara sederhana, fungsi tujuan dapat dituliskan sebagai berikut:

$$F = \max_{e \in E} load(e)$$

Fungsi tujuan dari model ini adalah meminimalkan nilai F , dengan e menyatakan tautan dalam jaringan, E menyatakan himpunan seluruh tautan, dan $load(e)$ menyatakan jumlah *flow* yang melewati tautan e . Dengan fungsi tujuan tersebut, algoritma tidak hanya mencari rute yang valid, tetapi juga berusaha menyebarkan aliran komunikasi agar tidak terlalu banyak menumpuk pada satu tautan tertentu. Biaya penyesuaian topologi OXC dipandang sebagai pertimbangan tambahan, sedangkan fokus utama penyelesaian diarahkan pada penurunan nilai *maximum flow conflict* menggunakan pendekatan *greedy*.

Selain meminimalkan *maximum flow conflict*, algoritma juga mempertimbangkan biaya penyesuaian topologi OXC sebagai kriteria tambahan. Pertimbangan ini digunakan ketika terdapat beberapa kandidat rute dengan nilai konflik yang sama. Dalam

kondisi tersebut, algoritma lebih memprioritaskan rute yang menggunakan koneksi OXC yang sudah tersedia atau berasal dari konfigurasi sebelumnya, sehingga perubahan topologi dapat dikurangi. Dengan demikian, fokus utama tetap berada pada penurunan nilai *maximum flow conflict*, sedangkan biaya penyesuaian topologi digunakan sebagai faktor pendukung dalam pemilihan rute.

IV. RANCANGAN ALGORITMA *GREEDY*

A. Ide Dasar Pendekatan *Greedy*

Pendekatan *greedy* digunakan untuk menentukan rute setiap *flow* secara bertahap berdasarkan kondisi jaringan saat pemilihan dilakukan. Setiap *flow* diproses satu per satu, kemudian algoritma mengevaluasi beberapa kandidat rute yang memungkinkan, baik dari koneksi yang sudah terbentuk pada *query* berjalan, koneksi dari konfigurasi sebelumnya, maupun koneksi baru yang dibentuk apabila diperlukan.

Keputusan lokal terbaik ditentukan berdasarkan beban tautan yang akan dilalui oleh kandidat rute. Kandidat dengan beban maksimum terkecil dipilih agar penumpukan aliran pada satu tautan dapat dikurangi. Jika terdapat beberapa kandidat dengan nilai beban maksimum yang sama, algoritma memprioritaskan rute yang lebih hemat perubahan topologi OXC, kemudian mempertimbangkan total beban rute sebagai kriteria berikutnya.

Dengan strategi ini, solusi dibangun secara inkremental tanpa mengevaluasi ulang rute yang telah dipilih sebelumnya. Oleh karena itu, pendekatan ini tidak menjamin solusi optimal secara global. Namun, algoritma ini sesuai digunakan sebagai heuristik karena mampu menghasilkan keputusan perutean secara cepat dengan tetap mempertimbangkan tujuan utama, yaitu meminimalkan *maximum flow conflict*.

B. Pengurutan *Flow*

Urutan pemrosesan *flow* berpengaruh terhadap kualitas solusi yang dihasilkan oleh algoritma *greedy*. Hal ini terjadi karena setiap rute yang telah dipilih akan memperbarui beban tautan dan konfigurasi OXC sehingga memengaruhi pilihan rute untuk *flow* berikutnya. Jika *flow* diproses langsung berdasarkan urutan masukan, terdapat kemungkinan beberapa *flow* dari pasangan *Group* yang sama diproses secara berurutan dan menggunakan sumber daya jaringan lebih awal.

Untuk mengurangi risiko tersebut, algoritma terlebih dahulu mengelompokkan *flow* berdasarkan pasangan *Group* asal dan tujuan (g_A, g_B) . Setelah pengelompokan dilakukan, *flow* diproses menggunakan strategi *round-robin*, yaitu mengambil satu *flow* dari setiap pasangan *Group* secara bergantian selama masih tersedia. Dengan cara ini, pemrosesan *flow* menjadi lebih tersebar dan tidak didominasi oleh satu pasangan *Group* tertentu.

Algoritma 1. Pengurutan *Flow* Menggunakan Round-Robin

Masukan: Daftar *flow* F , jumlah *Group* N
Keluaran: Urutan pemrosesan *flow_order*

- 1: Inisialisasi *pair_flows* berdasarkan pasangan *Group* (g_A, g_B)
 - 2: *flow_order* $\leftarrow \emptyset$
 - 3: *added* $\leftarrow \text{True}$
-

```

4: while added do
5:   added ← False
6:   for i ← 0 to N - 1 do
7:     for j ← 0 to N - 1 do
8:       if pair_flows[i][j] tidak kosong then
9:         ambil satu flow dari pair_flows[i][j]
10:        masukkan flow ke flow_order
11:        hapus flow tersebut dari
          pair_flows[i][j]
12:        added ← True
13:      end if
14:    end for
15:  end for
16: end while
17: return flow_order

```

Strategi pengurutan ini bertujuan membantu distribusi penggunaan tautan dan OXC menjadi lebih merata sejak awal proses perutean. Meskipun tidak menjamin penurunan konflik secara optimal, pengurutan *round-robin* dapat mendukung pendekatan *greedy* agar tidak terlalu cepat membebani sumber daya jaringan pada pasangan *Group* tertentu.

C. Pembentukan Kandidat Rute

Untuk setiap *flow*, algoritma membentuk kandidat rute dalam bentuk (x, k_x, m, y, k_y) . Kandidat tidak dihasilkan secara bebas dari seluruh kemungkinan, tetapi dibentuk bertahap agar pencarian tetap efisien. Tahap pertama memanfaatkan koneksi yang sudah terbentuk pada *query* berjalan. Tahap kedua mencoba menggunakan kembali koneksi dari konfigurasi sebelumnya. Tahap ketiga membentuk koneksi baru melalui *random search* terbatas, kemudian menggunakan pencarian cadangan jika belum ditemukan kandidat valid.

Urutan ini membuat algoritma tetap berfokus pada rute yang valid dan rendah konflik, sekaligus mengurangi perubahan topologi OXC apabila masih terdapat koneksi lama yang dapat digunakan.

Algoritma 2. Pembentukan Kandidat Rute

Masukan: Flow (gA, lA, gB, lB), konfigurasi OXC saat ini, konfigurasi OXC sebelumnya
 Keluaran: Himpunan kandidat rute

```

1: kandidat ← ∅
2: best_max_L ← ∞
3: for conn ∈ current_connections[gA][gB] do
4:   kandidat ← kandidat ∪ {conn}
5:   best_max_L ← min(best_max_L,
     Evaluate(conn))
6: end for
7: for conn ∈ prev_connections[gA][gB] do
8:   if IsFree(conn, config) then
9:     kandidat ← kandidat ∪ {conn}
10:    best_max_L ← min(best_max_L,
      Evaluate(conn))
11:   end if
12: end for
13: if best_max_L > 0 then
14:   for t ← 1 to K_SAMPLES do
15:     m ← RandomOXC()
16:     x ← RandomSpine(m)
17:     y ← RandomSpine(m)
18:     kx ← RandomLink()
19:     ky ← RandomLink()
20:     pA ← get_port(gA, x, kx)
21:     pB ← get_port(gB, y, ky)
22:     if config[m][pA] = -1 and
       config[m][pB] = -1 then
23:       kandidat ← kandidat ∪ {(m, pA,
         pB, x, kx, y, ky)}

```

```

24:       end if
25:     end for
26:   end if
27: end for
28: if kandidat = ∅ then
29:   kandidat ← FallbackSearch(gA, gB, config)
30: end if
31: return kandidat

```

D. Kriteria Pemilihan Rute

Setelah kandidat rute terbentuk, algoritma memilih satu rute terbaik untuk *flow* yang sedang diproses. Pemilihan dilakukan secara *greedy* berdasarkan kondisi beban jaringan saat itu. Setiap kandidat dievaluasi dengan memperhatikan empat komponen beban, yaitu beban dari *Leaf* asal ke *Spine* asal, beban dari *Leaf* tujuan ke *Spine* tujuan, serta beban pada dua port OXC yang digunakan.

Kriteria utama yang digunakan adalah nilai beban maksimum pada kandidat rute. Kandidat dengan beban maksimum terkecil dipilih karena dianggap paling mampu mengurangi potensi *maximum flow conflict*. Jika terdapat beberapa kandidat dengan nilai beban maksimum yang sama, algoritma memilih kandidat dengan tipe koneksi yang lebih diprioritaskan. Urutan prioritas tersebut adalah koneksi yang sudah terbentuk pada *query* berjalan, koneksi dari konfigurasi sebelumnya, lalu koneksi baru. Jika kedua kriteria tersebut masih sama, algoritma memilih kandidat dengan total beban terkecil.

Algoritma 3. Evaluasi Kandidat Rute

Masukan: Kandidat rute, beban jaringan, tipe koneksi
 Keluaran: Kandidat terbaik

```

1: L_max ← max(load_LS[gA][lA][x],
2:   load_LS[gB][lB][y],
3:   load_port[m][pA],
4:   load_port[m][pB])
5: L_sum ← load_LS[gA][lA][x]
6:   + load_LS[gB][lB][y]
7:   + load_port[m][pA]
8:   + load_port[m][pB]
9: if L_max < best_max_L then
10:   update kandidat terbaik
11: else if L_max = best_max_L and type <
   best_type then
12:   update kandidat terbaik
13: else if L_max = best_max_L and type =
   best_type and L_sum < best_sum_L then
14:   update kandidat terbaik
15: end if

```

E. Langkah Algoritma dan Kompleksitas

Secara umum, algoritma bekerja dengan memproses setiap *query* secara berurutan. Pada setiap *query*, seluruh *flow* terlebih dahulu diurutkan menggunakan strategi *round-robin*. Setelah itu, algoritma memproses *flow* satu per satu dengan membentuk kandidat rute, mengevaluasi kandidat berdasarkan beban jaringan, memilih rute terbaik, lalu memperbarui konfigurasi OXC dan beban tautan.

Setelah seluruh *flow* pada satu *query* selesai diproses, algoritma melakukan tahap pascapemrosesan dengan memasang kembali koneksi dari konfigurasi sebelumnya apabila kedua porta masih kosong. Tahap ini bertujuan mengurangi perubahan topologi OXC tanpa mengganggu rute yang sudah dipilih.

Konfigurasi hasil akhir kemudian disimpan sebagai konfigurasi sebelumnya untuk digunakan pada *query* berikutnya.

Algoritma 4. Langkah Utama Algoritma *Greedy*

Masukan: Daftar *query*, konfigurasi awal OXC

Keluaran: Konfigurasi OXC dan rute setiap *flow*

```

1:  prev_config ← konfigurasi kosong
2:  for setiap query do
3:    baca seluruh flow
4:    flow_order ← RoundRobinOrder(flow)
5:    config ← konfigurasi kosong
6:    inisialisasi load_LS dan load_port
7:    bentuk prev_connections dari prev_config
8:    for f ∈ flow_order do
9:      kandidat ← GenerateCandidate(f,
                                   config, prev_connections)
10:     route ← SelectBestRoute(kandidat,
                              load_LS, load_port)
11:     pasang route ke config
12:     perbarui load_LS dan load_port
13:     simpan route sebagai jawaban flow f
14:   end for
15:   RestoreUnusedPreviousConnections(config,
                                     prev_config)
16:   keluarkan config dan seluruh route
17:   prev_config ← config
18: end for

```

Kompleksitas algoritma dipengaruhi oleh jumlah *flow* pada setiap *query* dan jumlah kandidat rute yang diperiksa untuk setiap *flow*. Jika banyaknya *flow* dalam satu *query* dinyatakan sebagai Q , maka pemrosesan utama dilakukan sebanyak Q kali. Untuk setiap *flow*, algoritma mengevaluasi koneksi yang sudah ada, koneksi dari konfigurasi sebelumnya, serta sejumlah sampel acak yang dibatasi oleh konstanta $K_{SAMPLES}$.

Dalam implementasi ini, pencarian acak dibatasi sebanyak $K_{SAMPLES} = 1000$, sehingga waktu pencarian kandidat tetap terkendali. Secara praktis, kompleksitas per *query* dapat dipandang sebagai $O(Q \cdot C)$, dengan C menyatakan jumlah kandidat yang dievaluasi untuk setiap *flow*. Karena jumlah sampel acak dibatasi, nilai C tidak tumbuh sebesar seluruh kemungkinan rute.

Pada kondisi tertentu, apabila pencarian acak tidak menemukan kandidat valid, algoritma menjalankan pencarian cadangan secara menyeluruh hingga ditemukan pasangan port kosong. Dalam kasus terburuk, bagian ini dapat menambah biaya komputasi karena harus memeriksa kombinasi OXC, *Spine*, dan tautan. Namun, pencarian cadangan hanya digunakan sebagai mekanisme pengaman agar solusi tetap valid. Oleh karena itu, pendekatan ini tetap lebih efisien dibandingkan mengevaluasi seluruh kemungkinan rute untuk setiap *flow* sejak awal.

V. IMPLEMENTASI DAN EKSPERIMEN

Algoritma yang telah dirancang diimplementasikan menggunakan bahasa C++. Program membaca parameter jaringan dan daftar *flow* pada setiap *query*, kemudian menghasilkan keluaran berupa konfigurasi OXC dan rute untuk setiap *flow*. Keluaran program mengikuti format pada permasalahan, yaitu setiap *query* menghasilkan M baris konfigurasi OXC dan Q baris rute *flow*. Dengan demikian, program utama tidak langsung mencetak nilai *maximum flow conflict*, biaya penyesuaian topologi, maupun skor akhir.

Eksperimen dilakukan menggunakan lima *test case* dari *OXC Ports Arrangement Problem*. Kelima *test case* tersebut memiliki tingkat kompleksitas yang berbeda, mulai dari jaringan kecil pada *test case* 01 hingga jaringan besar pada *test case* 04 dan 05. Perbedaan ini digunakan untuk melihat kemampuan algoritma dalam menangani ukuran jaringan dan jumlah *flow* yang semakin besar.

Tabel 1. Karakteristik *test case*

Parameter	TC 01	TC 02	TC 03	TC 04	TC 05
N	2	4	4	32	32
S	1	2	2	16	16
L	2	2	4	30	32
M	2	2	2	240	256
K	1	1	2	1	2
P	1	1	1	8	16
R	2	8	16	64	64
Jumlah Query	5	5	5	5	5
Total Flow	10	40	71	38352	40912
Karakteristik Jaringan	Sangat Kecil	Kecil	Menengah	Besar	Besar

Untuk mengevaluasi hasil keluaran *solver*, digunakan program perhitungan skor. Program ini membaca kembali file masukan dan file keluaran *solver*, kemudian memeriksa konfigurasi OXC serta rute yang dihasilkan. Dari hasil tersebut, program menghitung nilai *maximum flow conflict*, biaya penyesuaian topologi OXC, skor akhir, dan waktu eksekusi. Dengan adanya program ini, keluaran *solver* yang semula hanya berupa rute dan konfigurasi dapat diubah menjadi data evaluasi yang dapat dianalisis.

Perhitungan *maximum flow conflict* dilakukan dengan menghitung jumlah *flow* yang melewati setiap tautan. Untuk setiap *flow*, program skor menelusuri jalur yang dipilih, yaitu *Leaf* asal ke *Spine* asal, *Spine* asal ke OXC, OXC ke *Spine* tujuan, dan *Spine* tujuan ke *Leaf* tujuan. Nilai konflik maksimum diperoleh dari beban terbesar di antara seluruh tautan tersebut.

Biaya penyesuaian topologi OXC dihitung dengan membandingkan konfigurasi OXC pada *query* saat ini terhadap konfigurasi pada *query* sebelumnya. Jika terdapat koneksi baru yang dibentuk, biaya bertambah satu. Jika terdapat koneksi lama yang dihapus, biaya juga bertambah satu. Pada *query* pertama, konfigurasi sebelumnya dianggap kosong karena belum ada koneksi OXC yang terbentuk.

Skor akhir dihitung mengikuti formula pada soal, yaitu:

$$Score_q = \frac{\alpha}{F_q \times C} + \beta \left(1 - \frac{A_q}{M \times R} \right)$$

dengan F_q menyatakan *maximum flow conflict* pada *query* ke- q , C menyatakan *convergence ratio*, A_q menyatakan biaya penyesuaian topologi OXC, M menyatakan jumlah OXC, dan R menyatakan jumlah port pada setiap OXC. Nilai konstanta yang digunakan adalah $\alpha = 1000$ dan $\beta = 300$. Nilai konstanta ini digunakan karena mengikuti formula skor resmi pada *OXC Ports Arrangement Problem*. Konstanta α memberikan bobot pada komponen *maximum flow conflict*, sedangkan β memberikan bobot pada komponen biaya penyesuaian topologi OXC. Nilai α yang lebih besar menunjukkan bahwa penurunan konflik aliran menjadi prioritas utama dalam penilaian, sementara biaya penyesuaian OXC tetap diperhitungkan sebagai

faktor pendukung. Skor total satu *test case* diperoleh dari penjumlahan skor seluruh *query* pada *test case* tersebut.

Tabel 2. Hasil eksperimen *solver*

Test Case	Rata-rata Maximum Flow Conflict	Maximum Flow Conflict Tertinggi	Total Biaya Penyesuaian OXC	Skor Total	Waktu Eksekusi (ms)
01	1	1	2	6350	294.98
02	2	2	12	3775	13.012
03	1.8	2	57	3965.625	16.377
04	2	2	36146	3294.023	792.089
05	2	2	31963	3414.74	757.008

Berdasarkan tabel hasil, analisis dilakukan dengan memperhatikan tiga hal utama. Pertama, nilai *maximum flow conflict* digunakan untuk menilai kemampuan algoritma dalam menyebarkan beban komunikasi. Kedua, biaya penyesuaian OXC digunakan untuk melihat seberapa besar perubahan topologi yang terjadi antar-*query*. Ketiga, waktu eksekusi digunakan untuk melihat efisiensi algoritma ketika ukuran jaringan dan jumlah *flow* meningkat. Dengan demikian, eksperimen ini tidak hanya menilai validitas rute yang dihasilkan, tetapi juga mengevaluasi kualitas solusi dari sisi konflik jaringan, stabilitas konfigurasi, dan efisiensi komputasi.

VI. ANALISIS

Berdasarkan hasil eksperimen, algoritma *greedy* mampu menghasilkan nilai *maximum flow conflict* yang rendah pada seluruh *test case*. Nilai konflik tertinggi tidak melebihi 2, sehingga dapat dikatakan bahwa strategi pemilihan rute berdasarkan beban maksimum kandidat cukup efektif dalam menyebarkan *flow* agar tidak terlalu menumpuk pada satu tautan. TC 01 menghasilkan konflik paling kecil, yaitu 1 karena ukuran jaringan dan jumlah *flow* masih sangat sederhana.

Pada TC 02, rata-rata konflik meningkat menjadi 2 karena jumlah *Group* dan *flow* lebih besar dibandingkan TC 01. TC 03 menunjukkan hasil sedikit lebih baik dengan rata-rata konflik 1.8. Hal ini disebabkan oleh nilai ($K = 2$), sehingga tersedia lebih banyak pilihan tautan antara *Spine* dan OXC. Dengan pilihan koneksi yang lebih banyak, algoritma memiliki ruang lebih besar untuk menyebarkan beban komunikasi.

Pada TC 04 dan TC 05, rata-rata konflik berada pada nilai 2 meskipun jumlah *flow* jauh lebih besar. Hasil ini menunjukkan bahwa pendekatan *greedy* masih mampu menjaga konflik tetap rendah pada jaringan besar. Namun, biaya penyesuaian topologi OXC meningkat tajam karena jumlah OXC, port, dan *flow* yang diproses juga jauh lebih besar. TC 05 memiliki biaya penyesuaian lebih rendah daripada TC 04 karena nilai ($K = 2$) dan jumlah OXC lebih besar, sehingga pilihan koneksi lebih banyak dan perubahan topologi dapat lebih ditekan. beban tautan.

Dari sisi skor, TC 01 memperoleh skor tertinggi karena konflik rendah dan biaya penyesuaian kecil. Skor TC 04 dan TC 05 lebih rendah karena skala kasus jauh lebih besar dan biaya penyesuaian topologi meningkat. Namun, TC 05 memiliki skor lebih baik daripada TC 04 karena biaya penyesuaiannya lebih rendah.

Waktu eksekusi menunjukkan bahwa algoritma cukup efisien. TC 02 dan TC 03 selesai dalam belasan milidetik,

sedangkan TC 04 dan TC 05 masih dapat diselesaikan dalam waktu kurang dari satu detik. Hal ini menunjukkan keunggulan algoritma *greedy*, yaitu tidak mengevaluasi seluruh kemungkinan konfigurasi secara global, melainkan mengambil keputusan lokal yang cepat.

Secara umum, algoritma *greedy* cocok digunakan sebagai heuristik untuk permasalahan konfigurasi dan perutean OXC. Algoritma ini tidak menjamin solusi optimal secara global karena keputusan awal dapat memengaruhi pilihan rute berikutnya dan berpotensi menimbulkan *bottleneck*. Namun, untuk kebutuhan *real-time*, pendekatan ini relevan karena mampu menghasilkan solusi valid dengan waktu eksekusi yang cepat dan nilai konflik yang relatif rendah.

VII. KESIMPULAN DAN SARAN

Berdasarkan hasil eksperimen, algoritma *greedy* dapat digunakan sebagai pendekatan heuristik untuk menyelesaikan permasalahan konfigurasi dan perutean OXC. Dengan memilih rute berdasarkan beban jaringan saat itu, algoritma mampu menjaga nilai *maximum flow conflict* tetap rendah, yaitu tidak lebih dari 2 pada seluruh *test case*. Selain itu, waktu eksekusi yang relatif kecil menunjukkan bahwa pendekatan ini cukup efisien untuk kebutuhan perutean cepat.

Namun, algoritma *greedy* tetap memiliki keterbatasan karena keputusan dibuat secara lokal. Rute yang dipilih pada awal proses dapat memengaruhi pilihan rute berikutnya dan belum tentu menghasilkan solusi optimal secara global. Oleh karena itu, pendekatan ini lebih tepat digunakan sebagai *baseline* atau strategi awal.

Sebagai pengembangan, hasil dapat ditingkatkan dengan menambahkan metode *local search* untuk memperbaiki rute setelah solusi awal terbentuk. Selain itu, variasi *randomization* atau *metaheuristic* juga dapat digunakan agar algoritma memiliki peluang menemukan konfigurasi dengan konflik dan biaya penyesuaian OXC yang lebih rendah.

LINK GITHUB

<https://github.com/NikSamSim/oxc-greedy-routing>

UCAPAN TERIMA KASIH

Segala puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa atas rahmat dan kekuatan yang telah diberikan, sehingga penulis dapat menyelesaikan makalah ini sebagai bagian dari proses pembelajaran di semester keempat. Penulis juga menyampaikan terima kasih yang sebesar-besarnya kepada keluarga tercinta yang senantiasa memberikan dukungan moril, semangat, dan doa di setiap langkah perjalanan akademik ini. Ucapan terima kasih yang tulus juga penulis sampaikan kepada dosen pengampu mata kuliah IF2211 Strategi Algoritma K01, Bapak Dr. Ir. Rinaldi, M.T., atas bimbingan, materi yang inspiratif, dan ruang eksplorasi yang beliau berikan sepanjang perkuliahan. Tak lupa, apresiasi setinggi-tingginya juga penulis berikan kepada teman-teman seperjuangan yang selalu saling membantu, berbagi semangat, dan menjadi tempat bertukar pikiran di tengah segala tantangan yang dihadapi. Semoga segala kebaikan dan dukungan yang telah diberikan mendapatkan balasan yang setimpal.

REFERENSI

- [1] R. Munir, "Algoritma Greedy (Bagian 1)," Informatika STEI ITB, Bandung, Indonesia. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/04-Algoritma-Greedy-\(2026\)-Bag1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/04-Algoritma-Greedy-(2026)-Bag1.pdf). [Accessed: Jun. 1, 2026].
- [2] R. Munir, "Algoritma Greedy (Bagian 2)," Informatika STEI ITB, Bandung, Indonesia. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/05-Algoritma-Greedy-\(2026\)-Bag2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/05-Algoritma-Greedy-(2026)-Bag2.pdf). [Accessed: Jun. 1, 2026].
- [3] R. Munir, "Algoritma Greedy (Bagian 3)," Informatika STEI ITB, Bandung, Indonesia. [Online]. Available: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/06-Algoritma-Greedy-\(2026\)-Bag3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/06-Algoritma-Greedy-(2026)-Bag3.pdf). [Accessed: Jun. 1, 2026].
- [4] Codeforces, "A. OXC Ports Arrangement Problem," ICPC 2025 Online Winter Challenge powered by Huawei. [Online]. Available: <https://codeforces.com/contest/2177/problem/A>. [Accessed: Jun. 1, 2026].
- [5] T. Ishikawa, Y. Mori, H. Hasegawa, S. Subramaniam, K. Sato, and O. Moriwaki, "Compact OXC architecture, design and prototype development for flexible waveband routing optical networks," Optics Express, vol. 25, no. 14, pp. 15838–15853, 2017, doi: 10.1364/OE.25.015838.

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, 19 Juni 2026



Niko Samuel Simanjuntak
13524029