Nutrifood Transporter Routing Optimization Problem Menggunakan Spiral Dynamic Optimization Algorithm

Dokumentasi Modelling dan Penyelesaian
Menggunakan R

Departemen Market Research
Nutrifood Indonesia

17 January 2023

Contents

List of Tables

List of Figures

1 PENDAHULUAN

1.1 Latar Belakang

Setiap hari, tim DTA membuat rute untuk *transporter* mendistribusikan produk jadi ke konsumen-konsumen Nutrifood yang telah melakukan *order*. Proses ini masih dilakukan secara manual. Akibatnya proses ini memakan waktu yang cukup lama dan tidak ada kejaminan bahwa rute yang dipilih sudah optimal atau belum. Oleh karena itu, tim DTA bersama dengan tim *Digital Transformation* dan *Market Research* berusaha untuk membuat model optimisasi dari permasalahan ini.

1.2 Tujuan

Membuat model optimisasi rute *transporter* yang meminimalkan *total cost* yang dibuat.

1.3 Ruang Lingkup

Business process yang terjadi selama ini sangat kompleks, oleh karena itu penelitian ini dibatasi pada lingkup sebagai berikut saja:

***Business Process* yang Hendak Dikerjakan**

Untuk mengirimkan produk jadi dari Gudang Ciawi dan Cibitung, tim DTA menyewa *transporter* dengan berbagai jenis armada kendaraan. Masing-masing kendaraan tersebut memiliki spesifikasi yang berbeda-beda, seperti:

1. Kapasitas maksimal kubikasi yang bisa diangkut,
2. Kapasitas maksimal tonase yang bisa diangkut,
3. Biaya sewa (per km). Diasumsikan biaya sewa ini nilainya tetap (tidak dipengaruhi oleh faktor lain seperti *habit* supir dan perbedaan rute yang ditempuh), dan
4. *Loading time*.

Masing-masing armada tersebut juga memiliki keterbatasan dari segi jumlah armada yang bisa disewa dan berapa banyak titik konsumen yang bisa dilalui.

Konsumen memesan (melalui proses *purchase order* - PO) sejumlah produk jadi kepada Nutrifood. Pada PO tersebut, kita memiliki informasi sebagai berikut:

1. Berapa total kubik dan tonase produk yang harus dikirim.
2. *Range* tanggal pengiriman produk.

Nutrifood harus memenuhi pembelian tersebut secara langsung (tidak boleh memecah pengiriman produk dalam satu PO menjadi beberapa kali pengiriman). Masing-masing konsumen akan dilayani oleh gudang Ciawi atau Cibitung sesuai dengan pembagian yang telah ditetapkan sebelumnya. Tidak ada konsumen yang dilayani oleh keduanya.

Masing-masing konsumen memiliki keterbatas lain terkait armada yang bisa dilalui karena lokasi mereka berbeda-beda. Ada konsumen yang berlokasi di jalan besar sehingga armada

ukuran besar bisa melewatinya dengan aman. Namun ada beberapa konsumen yang lokasinya hanya bisa dilalui oleh armada kecil.

1.4 Metode Penyelesaian Model Optimisasi

Untuk menyelesaikan model optimisasi ini, saya akan menggunakan pendekatan *meta heuristic* dibandingkan penyelesaian secara *exact*. Berikut alasannya:

1. Kita tidak perlu menuliskan model matematika yang kompleks karena permasalahan yang kita hadapi ini memiliki indeks yang tinggi. Kita cukup menuliskan algoritma (*computational model*) berdasarkan definisi dan *constraints* secara logis.
2. Penyelesaian dengan metode *exact* memang menjamin keoptimalan solusi namun tidak semua model bisa dicari solusinya. Sedangkan penyelesaian dengan metode *meta heuristic*, tidak menjamin solusi yang didapatkan adalah solusi yang paling optimal. Namun kita bisa menjadikan solusi tersebut *near optimal* dengan melakukan *tweaking* pada algoritma.

Pendekatan *meta heuristic* yang akan saya gunakan adalah *Spiral Dynamic Optimization Algorithm*.

2 SDOA

2.1 Penjelasan Singkat

Spiral Dynamic Optimization Algorithm (SDOA) adalah salah satu metode *meta heuristic* yang digunakan untuk mencari minimum global dari suatu sistem persamaan.

Algoritmanya mudah dipahami dan intuitif tanpa harus memiliki latar keilmuan tertentu. Proses kerjanya adalah dengan melakukan *random number generating* pada suatu selang dan melakukan rotasi sekaligus kontraksi dengan titik paling minimum pada setiap iterasi sebagai pusatnya.

Berikut adalah algoritmanya:

```
INPUT
  m >= 2 # jumlah titik
  theta  # sudut rotasi (0 <= theta <= 2pi)
  r      # konstraksi
  k_max  # iterasi maksimum
PROCESS
  1 generate m buah titik secara acak
      x_i
  2 initial condition
      k = 0 # untuk keperluan iterasi
  3 cari x_* yang memenuhi
      min(f(x_*))

  4 lakukan rotasi dan konstraksi semua x_i
      x_* sebagai pusat rotasi
      k = k + 1
  5 ulangi proses 3 dan 4
  6 hentikan proses saat k = k_max
      output x_*
```

Berdasarkan algoritma di atas, salah satu proses yang penting adalah melakukan **rotasi** dan **konstraksi** terhadap semua titik yang telah di-*generate*.

2.2 Menyelesaikan Masalah Optimisasi dengan SDOA

Misal suatu permasalahan MILP atau MINLP bisa ditulis secara umum sebagai berikut:

$$\min_{x \in \mathbb{R}^n} f(x)$$

$$\text{subject to: } g_i(x) = 0, i = 1, 2, .., M$$

$$\text{and } h_j(x) \leq 0, i = 1, 2, .., N$$

$$x = (x_1, x_2, ..., x_n)^T \in \mathbb{N}$$

Bentuk di atas bisa kita ubah menjadi:

$$F(x, \alpha, \beta) = f(x) + \sum_{i=1}^{M} \alpha_i g_i^2(x) + \sum_{j=1}^{N} \beta_j (\max(h_i(x), 0))^2$$

dimana α, β merupakan *penalty constant* yang bisa dibuat sangat besar.

2.3 Matriks Rotasi untuk n-Dimensi

SOA relatif mudah untuk dituliskan dalam bentuk algoritma bahasa pemrograman manapun. Tapi ada satu hal yang bisa menjadi batu ganjalan dalam menuliskan algoritmanya. Apa itu? Yaitu pendefinisian matriks rotasi untuk masalah dengan n-dimensi.

Bentuk umum dari matriks rotasi adalah sebagai berikut:

$$R^{(n)}(\theta_{1,2}, \theta_{1,3}, .., \theta_{n,n-1}) = \prod_{i=1}^{n-1} \left(\prod_{j=1}^{i} R^{(n)}_{n-i,n+1-j}(\theta_{n-i,n+1-j}) \right)$$

Perhatikan bahwa perkalian matriks rotasi yang dilakukan adalah *cross product*.

Alasan: Rotasi tidak mengubah *norm* suatu vektor.

3 DATA TERKAIT

Data *real* dari DTA dan gudang sedang disusun oleh tim terkait. Oleh karena itu, saya akan gunakan data *dummy* berdasarkan informasi pada bagian sebelumnya.

3.1 Data Terkait *Order* Toko

Table 1: Data Order Toko

nama_toko	order_kubikasi	order_tonase	tanggal_kirim_min	tanggal_kirim_max
toko dominiqua	20	22.6	2	2
toko emilio	23	25.9	1	3
toko wisaam	6	6.8	1	3
toko hafsa	14	15.8	1	3
toko miguel	26	29.3	2	6
toko marisol	5	5.6	1	5
toko vyktoreya	19	21.4	1	4
toko meagan	5	5.6	2	3
toko nicholas	30	33.8	5	7
toko madeline	29	32.7	3	4
toko tiffanie	22	24.8	7	7
toko andrea	26	29.3	2	2
toko anatacia	22	24.8	5	6
toko adham	27	30.4	7	7
toko cierra	30	33.8	4	5
toko thaleia	22	24.8	1	2
toko elizabeth	15	16.9	1	3
toko amru	28	31.6	1	7
toko lizbeth	28	31.6	2	6
toko nizhoni	23	25.9	5	7

Penjelasan terkait variabel dari tabel di atas:

1. `nama_toko`, yakni nama-nama toko yang melakukan *order* produk ke Nutrifood.
2. `order_kubikasi`, yakni berapa total kubik produk yang dipesan. Satuan yang digunakan adalah m^2.
3. `order_tonase`, yakni berapa total kilogram produk yang dipesan.
4. `tanggal_kirim_min`, yakni tanggal berapa produk sudah bisa dikirim.
5. `tanggal_kirim_max`, yakni tanggal berapa produk paling lambat harus dikirim.

3.2 Data Terkait Informasi Detail Toko

Table 2: Data Informasi Toko

nama_toko	long	lat	max_armada	supplied
toko dominiqua	0.2346432	0.5126298	2	ciawi
toko emilio	0.3066549	0.9328371	2	ciawi
toko wisaam	0.8225916	0.5368827	5	ciawi
toko hafsa	0.6652651	0.6179265	2	ciawi
toko miguel	0.3114318	0.4979370	3	cibitung
toko marisol	0.2868179	0.0244884	1	cibitung
toko vyktoreya	0.5831267	0.2104877	3	ciawi
toko meagan	0.7970947	0.8795184	2	ciawi
toko nicholas	0.9436925	0.2113620	3	cibitung
toko madeline	0.6369165	0.3728857	3	ciawi
toko tiffanie	0.6158117	0.5142791	3	ciawi
toko andrea	0.3520580	0.5610708	4	ciawi
toko anatacia	0.4515903	0.2660503	2	cibitung
toko adham	0.4237576	0.8794421	5	ciawi
toko cierra	0.1558586	0.4834961	4	ciawi
toko thaleia	0.6333142	0.7290114	5	cibitung
toko elizabeth	0.2798145	0.3276253	4	ciawi
toko amru	0.0159862	0.3550737	5	ciawi
toko lizbeth	0.4959727	0.4719731	2	ciawi
toko nizhoni	0.4659126	0.0712972	3	ciawi

Penjelasan terkait variabel dari tabel di atas:

1. `nama_toko`, yakni nama-nama toko yang melakukan *order* produk ke Nutrifood.
2. `long`, yakni *longitude* dari alamat toko.
3. `lat`, yakni *latitude* dari alamat toko.
4. `max_armada`, yakni jenis armada terbesar yang bisa masuk ke toko. Misalkan, jika `max_armada = 2`, artinya toko tersebut bisa dilalui armada jenis `1` dan `2`.
5. `supplied`, yakni gudang yang men-*supply* toko tersebut.

3.3 Data Terkait Gudang

Table 3: Data Time Slot Gudang

site	week_day_hour	week_end_hour
ciawi	13.5	10
cibitung	13.5	10

Penjelasan terkait variabel pada tabel di atas:

1. `site`, jenis gudang: Ciawi atau Cibitung.
2. `week_day_hour`, total waktu kerja yang tersedia pada hari kerja untuk melakukan *loading* produk dari gudang ke armada. Satuan dari data ini adalah dalam jam.
3. `week_end_hour`, total waktu kerja yang tersedia pada hari libur untuk melakukan *loading* produk dari gudang ke armada. Satuan dari data ini adalah dalam jam.

Kedua data total waktu kerja ini berdasarkn jam kerja pada dua *shift*.

3.4 Data Terkait Armada

Table 4: Data Informasi Armada

armada	max_cap_kubikasi	max_cap_tonase	cost_per_km	tersedia	max_titik	loading_time
1	17	32.2	4.6	9	2	0.24
2	23	34.9	11.8	7	3	0.29
3	27	29.2	28.2	5	3	0.44
4	32	55.9	29.2	8	7	0.90
5	34	52.4	34.4	7	8	0.97

Penjelasan terkait variabel dari tabel di atas:

1. `armada`, yakni jenis armada yang bisa disewa Nutrifood.
2. `max_cap_kubikasi`, yakni kapasitas maksimum kubikasi yang bisa diangkut oleh armada tersebut. Satuan dari data ini m^2.
3. `max_cap_tonase`, yakni kapasitas maksimum berat barang yang bisa diangkut oleh armada tersebut. Satuan dari data ini kg.
4. `cost_per_km`, yakni berapa biaya sewa mobil per kilometer untuk mobil tersebut. Satuan dari data ini *Rp*. Informasi dari tim DTA:
   - Secara *real*, nilainya berbeda-beda tergantung *provider* yang digunakan walau jenis mobilnya sama.
   - Hal ini terjadi karena perbedaan *habit* mengemudi para *driver* dan rute yang diambil.
   - Oleh karena itu, pada kasus ini, nilainya kita asumsikan sama karena tidak ada kepastian *provider* mana yang akan tersedia pada hari pengiriman tersebut.
5. `tersedia`, yakni berapa banyak armada tersebut tersedia untuk disewa. Informasi dari tim DTA:
   - Pada kondisi *real*, tidak ada pembatasan berapa banyak armada yang tersedia. Bisa diasumsikan nilainya *unlimited*.
   - Namun, ada baiknya jika kita masukan parameter batas ini untuk mengakomodir kebutuhan di kemudian hari.
   - Untuk keperluan komputasi, ketersediaan ini tidak saya jadikan parameter pada model, tapi digunakan untuk mereplikasi baris data pada tabel di atas.
6. `max_titik`, yakni berapa banyak maksimal konsumen yang pesanannya bisa diantar.
7. `loading_time`, yakni berapa lama proses *loading* yang dibutuhkan untuk masing-masing armmada di gudang Ciawi atau Cibitung. Satuan dari data ini adalah jam.

4 *MATHEMATICAL MODEL*

Menuliskan model matematika dari permasalahan kompleks di atas menjadi tantangan tersendiri karena variabel yang terlibat akan memiliki indeks yang tinggi, setidaknya ada 4 indeks yang berasal dari 4 himpunan yang terlibat:

4.1 *Index* dan Himpunan yang Terlibat

- $\mathcal{T} = \{1, 2, .., t\}$ sebagai himpunan toko yang memesan produk ke Nutrifood.
- $\mathcal{M} = \{1, 2, .., m\}$ sebagai himpunan jenis armada yang bisa disewa Nutrifood.
- $\mathcal{G} = \{1, 2\}$ sebagai himpunan gudang yang men-*supply* semua toko yang ada.
- $\mathcal{D} = \{1, 2, .., d\}$ sebagai himpunan tanggal pengiriman produk dari Nutrifood ke toko.
  - $\hat{\mathcal{D}}$ sebagai hari *weekday*.
  - $\dot{\mathcal{D}}$ sebagai hari *weekend*.

4.2 Parameter dari Data

Tuliskan:

- $ok_t, t \in \mathcal{T}$ sebagai *order* kubikasi toko t.
- $ot_t, t \in \mathcal{T}$ sebagai *order* tonase toko t.
- $\forall t \in \mathcal{T}, tgl1_t$ sebagai tanggal minimal pengiriman produk oleh Nutrifood untuk toko t.
- $\forall t \in \mathcal{T}, tgl2_t$ sebagai tanggal maksimal pengiriman produk oleh Nutrifood untuk toko t.
- $\forall t_1, t_2 \in \mathcal{T}, J_{t_1 t_2}$ sebagai jarak antara toko t_1 dan toko t_2.
- $\forall m \in \mathcal{M}, maxcap1_m$ sebagai max kapasitas kubikasi yang bisa diangkut armada m.
- $\forall m \in \mathcal{M}, maxcap2_m$ sebagai max kapasitas tonase yang bisa diangkut armada m.
- $\forall m \in \mathcal{M}, cost_m$ sebagai biaya sewa perkilometer armada m.
- $\forall m \in \mathcal{M}, temp_m$ sebagai max banyaknya toko yang bisa diantarkan armada m.
- $\forall m \in \mathcal{M}, lt_m$ sebagai *loading time* armada m.
- $\forall g \in \mathcal{G}, ts1_g$ sebagai total *time slot* gudang g pada *weekday*.
- $\forall g \in \mathcal{G}, ts2_g$ sebagai total *time slot* gudang g pada *weekend*.

Dari *sets* dan parameter di atas, kita akan buat *computational model*-nya sebagai berikut.

5 *COMPUTATIONAL MODEL*

Untuk membuat model komputasinya, kita buat terlebih dahulu beberapa *function* dasar berikut ini:

5.1 *Function* Dasar

Berikut adalah beberapa *function* dasar yang akan digunakan pada SDOA.

5.1.1 *Rotation Matrix*

Ini adalah *function* untuk membuat matriks rotasi:

```
# function matriks rotasi
buat_rot_mat = function(theta,n){
  # buat template sebuah matriks identitas
  temp_mat = matrix(0,ncol = n,nrow = n)
  diag(temp_mat) = 1

  # buat matriks identitas terlebih dahulu
  mat_rot = temp_mat
  # membuat isi matriks rotasi
  for(i in 1:(n-1)){
    for(j in 1:i){
      temp = temp_mat
      idx = n-i
      idy = n+1-j
      # print(paste0("Matriks rotasi untuk ",idx," - ",idy,": DONE"))
      temp[idx,idx] = cos(theta)
      temp[idx,idy] = -sin(theta)
      temp[idy,idx] = sin(theta)
      temp[idy,idy] = cos(theta)
      # assign(paste0("M",idx,idy),temp)
      mat_rot = mat_rot %*% temp
      mat_rot = mat_rot
    }
  }
  # output matriks rotasi
  return(mat_rot)
}
```

5.1.2 *Generator* Calon Solusi

Kelak calon solusi yang diharapkan memiliki format sebagai berikut:

Table 5: Calon Solusi yang Diharapkan

nama_toko	order_kubikasi	order_tonase	armada_kirim	tanggal_kirim
toko dominiqua	20	22.6	1	1
toko emilio	23	25.9	2	3
toko wisaam	6	6.8	5	3
toko hafsa	14	15.8	3	1
toko miguel	26	29.3	1	3
toko marisol	5	5.6	3	4
toko vyktoreya	19	21.4	1	4
toko meagan	5	5.6	2	3
toko nicholas	30	33.8	5	7
toko madeline	29	32.7	5	3
toko tiffanie	22	24.8	4	4
toko andrea	26	29.3	3	1
toko anatacia	22	24.8	3	6
toko adham	27	30.4	4	6
toko cierra	30	33.8	5	4
toko thaleia	22	24.8	5	2
toko elizabeth	15	16.9	2	2
toko amru	28	31.6	1	6
toko lizbeth	28	31.6	1	3
toko nizhoni	23	25.9	2	6

Oleh karena itu, solusi yang perlu di-*generate* ada dua variabel:

1. `armada_kirim`, yakni armada yang digunakan untuk mengirimkan pesanan toko.
2. `tanggal_kirim`, yakni tanggal pesanan toko dikirim.

Berikut adalah *function generator*-nya:

```
# generate solusi untuk armada
armada_generate = function(n_toko,n_armada){
  sample(n_armada,n_toko,replace = T)
}

# generate tanggal kirim sesuai dengan data yang ada pada df_order
tanggal_generate = function(var,df){
  hasil = rep(0,n_toko)
  min   = df[["tanggal_kirim_min"]] %>% as.numeric()
  max   = df[["tanggal_kirim_max"]] %>% as.numeric()
  for(i in 1:n_toko){
```

```
    if(min[i] == max[i]){
      hasil[i] = min[i]
    }
    if(min[i] != max[i]){
      hasil[i] = sample(c(min[i]:max[i]),1)
    }
  }
  return(hasil)
}
```

5.1.3 Menghitung Matriks Jarak

Berikut adalah *function* untuk matriks jarak:

```
# function untuk membuat matriks jarak
buat_matriks_jarak = function(df){
  n_toko = nrow(df)
  # buat rumahnya terlebih dahulu
  dist_mat = matrix(0,n_toko,n_toko)
  # kita buat euclidean distance terlebih dahulu
  hitung_jarak = function(i,j){
    lon_hit = df$long[i] - df$long[j]
    lat_hit = df$lat[i] - df$lat[j]
    jarak = sqrt(lon_hit^2 + lat_hit^2)
    round(jarak,3)
  }
  # kita hitung jaraknya sekarang
  for(i in 1:n_toko){
    for(j in 1:n_toko){
      dist_mat[i,j] = hitung_jarak(i,j)
    }
  }
  # hasil finalnya
  return(dist_mat)
}
```

5.1.4 Menghitung Panjang Rute Optimal

Berikut adalah *function* untuk menghitung panjang rute optimal dari *input* berupa *database* toko:

```
# perhitungan rute optimal
# inputnya adalah matriks jarak
tsp_hitung = function(new){
  # jangan lupa new adalah df_toko yang sudah di-slice
  jarse = buat_matriks_jarak(new)
```

```
  problem = as.ATSP(jarse)
  hasil = solve_TSP(problem)
  level = row.names(new)
  panjang_rute = tour_length(hasil)
  detail_rute  = paste(level[as.integer(hasil_1)],collapse = " - ")
  return(panjang_rute)
}
```