Penyelesaian Nutrifood Transporter Routing Optimization Problem Menggunakan Spiral Dynamic Optimization Algorithm

CASE STUDIES

Dokumentasi Modelling dan Penyelesaian
Menggunakan R

Departemen Market Research
Nutrifood Indonesia

18 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*.

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. Sehingga proses *modelling* bisa dilakukan dengan lebih cepat.
2. Penyelesaian dengan metode *exact* memang menjamin keoptimalan solusi. Sedangkan penyelesaian dengan metode *meta heuristic*, walaupun tidak ada jaminan solusi yang didapatkan adalah solusi yang paling optimal tapi kita bisa mencari solusi *near optimal* dengan melakukan *tweaking* pada algoritma tersebut.

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. SDOA termasuk ke dalam salah satu *greedy algorithm* yang biasa digunakan untuk memecahkan berbagai masalah dalam sains dan *engineering*.

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. Pusat rotasi akan dipilih dari calon solusi yang *near optimal*. Dengan adanya iterasi tersebut, proses eksploitasi dan eksplorasi terhadap calon solusi lain bisa dijalankan sehingga kita akan dapatkan solusi paling optimal di akhir iterasi.

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

Permasalahan Nutrifood terkait *transporter* merupakan salah satu bentuk *Mixed Integer Non Linear Programming* (MINLP). Secara umum, bentuk MINLP bisa ditulis sebagai berikut:

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

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

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

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

Agar bisa diselesaikan dengan SDOA, kita akan mengubah bentuk di atas menjadi:

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

dimana α, β merupakan *penalty constant* yang bisa dibuat sangat besar. Model komputasi ini dikenal juga dengan sebutan *reinforce learning*.

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 al-bacchus, saeed	14	18.0	3	7
toko vialpando, jason	18	23.2	2	6
toko el-pashia, saabira	16	20.6	3	3
toko lefotu, jade	16	20.6	1	6
toko seubert, francesca	26	33.5	4	7
toko zulali, vanessa	25	32.2	2	7
toko mckinney, tyre	20	25.7	2	3
toko bell, morgan	17	21.9	2	5
toko coleman, kiana	24	30.9	1	2
toko gonzales, destiny	23	29.6	1	7
toko al-ahmed, hanoona	26	33.5	3	6
toko kellison, macella	10	12.9	5	6
toko martinez, londisha	13	16.7	2	6
toko leon, stephon	6	7.7	4	7
toko frye, lucas	22	28.3	2	5

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.

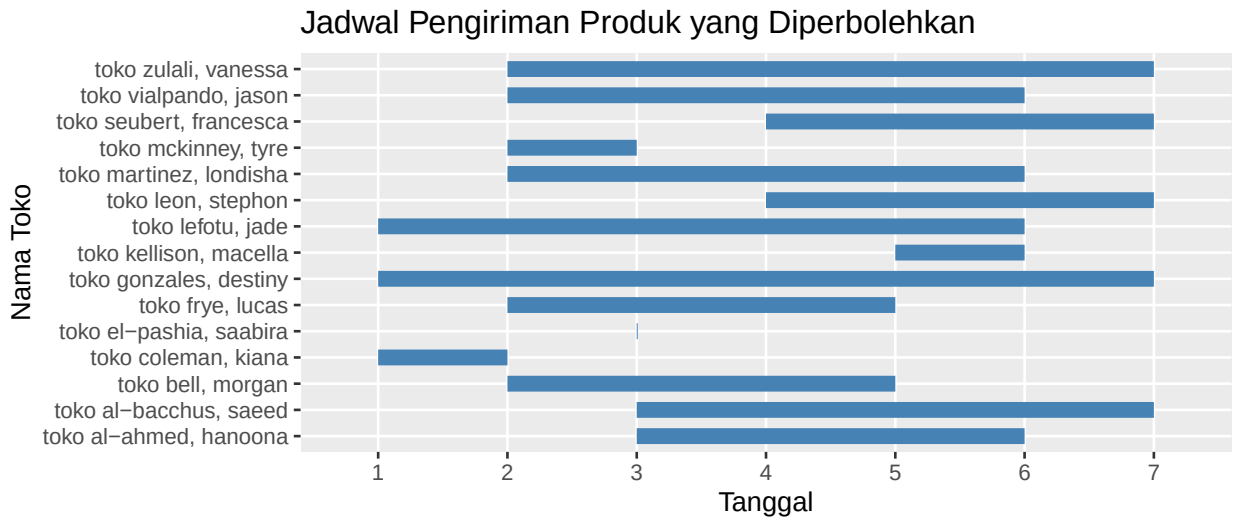

Figure 1: Jadwal Pengiriman yang Diperbolehkan

3.2 Data Terkait Informasi Detail Toko

Table 2: Data Informasi Toko

nama_toko	long	lat	max_armada	supplied
toko al-bacchus, saeed	0.9946344	0.3469710	1	ciawi
toko vialpando, jason	0.8676655	0.9181804	1	ciawi
toko el-pashia, saabira	0.0852901	0.0328836	1	ciawi
toko lefotu, jade	0.3708360	0.7901229	4	cibitung
toko seubert, francesca	0.2627471	0.5719211	4	cibitung
toko zulali, vanessa	0.7909945	0.1493886	3	ciawi
toko mckinney, tyre	0.4933766	0.9991480	2	ciawi
toko bell, morgan	0.3661260	0.1699563	5	ciawi
toko coleman, kiana	0.1267329	0.8324689	5	ciawi
toko gonzales, destiny	0.7145765	0.1323264	2	ciawi
toko al-ahmed, hanoona	0.7761427	0.0091447	5	ciawi
toko kellison, macella	0.9048722	0.8514034	3	ciawi
toko martinez, londisha	0.4217208	0.3262619	5	ciawi
toko leon, stephon	0.0515294	0.7946159	5	cibitung
toko frye, lucas	0.5938839	0.5670090	4	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.

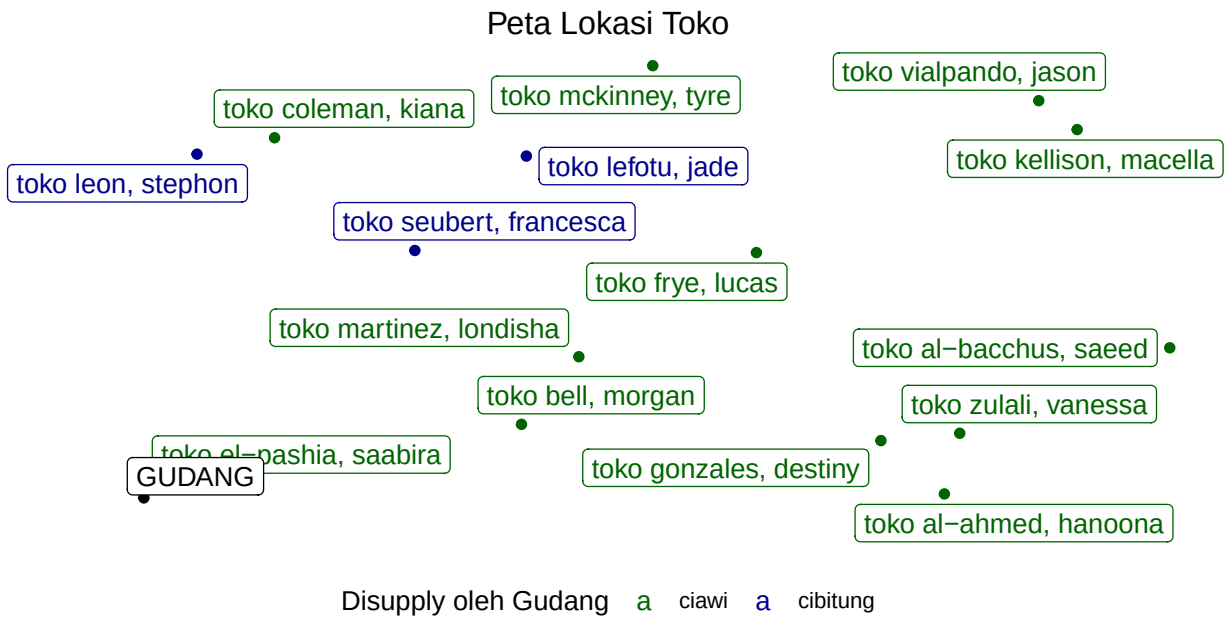

Figure 2: Peta Lokasi Toko

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 berdasarkan jam kerja pada dua *shift*.

3.4 Data Terkait Armada

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.

Table 4: Data Informasi Armada

armada	max_cap_kubikasi	max_cap_tonase	cost_per_km	tersedia	max_titik	loading_time
1	26	40.8	8.1	10	2	0.12
2	30	51.0	19.9	4	2	0.73
3	31	31.6	23.2	3	5	0.86
4	84	89.5	30.4	2	5	0.90
5	85	143.3	42.8	2	9	0.94

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*.

Oleh karena menuliskan model matematikanya rumit, maka informasi *sets* dan parameter yang ada di atas akan kita jadikan modal untuk menuliskan *computational model*-nya.

5 *COMPUTATIONAL MODEL*

Model komputasi yang digunakan saya akan lampirkan pada dokumen ini.

5.1 Asumsi

Beberapa asumsi yang digunakan:

1. Perhitungan jarak menggunakan formula *euclidean distance*, yakni panjang garis lurus antara dua titik. Kita dapatkan nilainya menggunakan hukum *Pytagoras* sederhana.
2. Total *loading time* gudang baru berisi satu nilai saja (belum ada pembagian *weekend* dan *weekday*).
3. Banyaknya ketersediaan armada berbeda-beda dan nilainya terbatas.
4. Model ini hanya meng-*cover* waktu pengiriman selama 7 hari.
5. Titik lokasi gudang dibuat dengan $long = 0$ dan $lat = 0$.

Parameter SDOA

Beberapa parameter SDOA yang digunakan:

1. $\theta = \frac{2\pi}{30}$,
2. Koefisien kontraksi = .7,
3. Banyak iterasi = 50,
4. Banyak calon solusi yang di-*generate* = 3000.

Model akan mencari solusi optimal dengan laju rotasi yang relatif cepat.

5.2 *Constraints*

Beberapa *constraints* yang ada pada model:

1. Tidak boleh ada armada yang kelebihan muatan dalam hal kubikasi dan tonase.
2. Banyaknya armada yang mengantar tidak boleh melebihi banyaknya ketersediaan armada.
3. Banyaknya toko yang ada pada rute tidak melebihi aturan maksimum toko per armada.
4. Waktu *loading* gudang perhari tidak boleh dilanggar.

Perhatikan bahwa semua bentuk *constraints* di atas adalah sebagai berikut:

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

5.3 *Objective Function*

Fungsi objektif pada model ini adalah:

> Meminimumkan total biaya yang timbul dari setiap keputusan toko i dikirim armada j pada tanggal k tanpa melanggar *constraints* yang ada.

5.4 Catatan Terkait *Libraries* yang Digunakan

Tidak ada *libraries* khusus yang digunakan. Semua algoritma ditulis menggunakan prinsip *tidyverse*.

5.5 Catatan Terkait Proses Komputasi

Namun untuk mempercepat komputasi, saya menggunakan prinsip *parallel computing* dengan banyaknya *cores* yang digunakan adalah 5.

Total *runtime* pada contoh kasus ini adalah sekitar 10 menit.

Oleh karena itu, algoritma ini hanya bisa di-*run* dengan komputer bersistem operasi Linux / UNIX. *Windows* secara *default* tidak *support* proses *parallel computing*.

6 SOLUSI MODEL

6.1 Pengiriman dari Gudang Ciawi

Berikut adalah solusi model untuk toko yang di-*supply* oleh gudang Ciawi:

Table 5: Pengiriman tanggal: 1

nama_toko	armada_ke	jenis_armada	order_kubikasi	order_tonase
toko gonzales, destiny	1	1	23	29.6

Table 6: Pengiriman tanggal: 2

nama_toko	armada_ke	jenis_armada	order_kubikasi	order_tonase
toko bell, morgan	1	1	17	21.9
toko el-pashia, saabira	2	1	16	20.6
toko zulali, vanessa	3	1	25	32.2

Table 7: Pengiriman tanggal: 3

nama_toko	armada_ke	jenis_armada	order_kubikasi	order_tonase
toko al-bacchus, saeed	1	1	14	18.0
toko kellison, macella	1	1	10	12.9
toko frye, lucas	2	1	22	28.3

Table 8: Pengiriman tanggal: 4

nama_toko	armada_ke	jenis_armada	order_kubikasi	order_tonase
toko al-ahmed, hanoona	1	1	26	33.5
toko vialpando, jason	2	1	18	23.2
toko coleman, kiana	3	1	24	30.9

Table 9: Pengiriman tanggal: 5

nama_toko	armada_ke	jenis_armada	order_kubikasi	order_tonase
toko mckinney, tyre	1	1	20	25.7
toko martinez, londisha	2	1	13	16.7

6.2 Pengiriman dari Gudang Cibitung

Berikut adalah solusi model untuk toko yang di-*supply* oleh gudang Cibitung:

Table 10: Pengiriman tanggal: 4

nama_toko	armada_ke	jenis_armada	order_kubikasi	order_tonase
toko lefotu, jade	1	1	16	20.6
toko leon, stephon	1	1	6	7.7

Table 11: Pengiriman tanggal: 6

nama_toko	armada_ke	jenis_armada	order_kubikasi	order_tonase
toko seubert, francesca	1	1	26	33.5

7 LAMPIRAN

Computational Model

Berikut adalah algoritma yang dibuat:

```
# membersihkan global environment
rm(list=ls())
setwd("~/NTOP/output")

# ================================================================================
# libraries yang diperlukan
library(dplyr)
library(tidyr)
library(TSP)

# ================================================================================
# load data
load("~/NTOP/input/data dokumentasi.rda")

# ================================================================================
# kita akan modifikasi si database jenis armada
# yakni dengan mereplikasi baris-baris tergantung dari ketersediaan armada
temp =
  df_jenis_armada %>%
  group_split(armada)

n_temp = length(temp)
df_hasil = data.frame()

for(i in 1:n_temp){
  df_temp = temp[[i]]
  for(k in 1:df_temp$tersedia){
    df_hasil = rbind(df_temp,df_hasil)
  }
}

# kita kembalikan lagi ke data awal
df_jenis_armada =
  df_hasil %>%
  arrange(armada) %>%
  mutate(id_armada = 1:nrow(df_hasil)) %>%
  select(-tersedia)
```

```
# lalu kita akan ambil data mana yang harus dikerjakan terlebih dahulu
# apakah mau Ciawi atau Cibitung terlebih dahulu?
# misalkan target gudang terlebih dahulu
target_gudang = "cibitung"

# kita filtering terlebih dahulu
df_toko   = df_toko %>% filter(supplied == target_gudang)
df_order  = df_order %>% filter(nama_toko %in% df_toko$nama_toko)
df_gudang = df_gudang %>% filter(site == target_gudang) %>% .$week_day_hour


# ================================================================================
# membuat function rotation matrix
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)
}


# ================================================================================
# berikutnya kita akan buat function generator
# jangan lupa bahwa nanti ada yang harus difilter terlebih dahulu
```

```
# 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)
}


# ================================================================================
# function untuk membuat matriks jarak
buat_matriks_jarak = function(df){
  n_toko = nrow(df)
  # kita tambahin untuk long lat kantor
  df[n_toko+1,] = list(NA)
  n_toko = nrow(df)
  df$long[n_toko] = 0
  df$lat[n_toko] = 0
  # 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)
}


# ================================================================================
# 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)],collapse = " - ")
  return(panjang_rute)
}


# ================================================================================
# kita akan buat objective function termasuk constraint di dalamnya
obj_func = function(list_1,list_2){
  # kita buat dulu ke data frame untuk mengecek semua informasi yang ada
  df_temp_1 = df_toko %>% select(nama_toko,long,lat,max_armada)
  df_temp_2 = df_order %>% select(nama_toko,order_kubikasi,order_tonase)
  df_temp_3 = merge(df_temp_1,df_temp_2) %>% distinct()
  df_temp_3$id_armada     = round(as.vector(list_1),0) # kita rounding dulu ya
  df_temp_3$tanggal_kirim = round(as.vector(list_2),0) # kita rounding dulu ya
  df_temp_3 = merge(df_temp_3,df_jenis_armada)

  # konstanta penalti
  # reinforce learning
  beta = 10^4

  # kita pecah dulu berdasarkan armada dan tanggal
  pecah      = df_temp_3 %>% group_split(id_armada,tanggal_kirim)
  n_pecah    = length(pecah)

  # kita hitung dulu cost per jarak
  jarak_cost = rep(0,n_pecah)
  for(i in 1:n_pecah){
    temp           = pecah[[i]]
    jarak_hit      = tsp_hitung(temp)
```

```
  jarak_cost[i]  = jarak_hit * temp$cost_per_km[1]
}
jarak_total  = sum(jarak_cost)   # ini yang pertama disave

# constraint 1
# tidak ada armada yang kelebihan muatan dalam kubikasi
constraint_1 = rep(0,n_pecah)
# constraint 2
# tidak ada armada yang kelebihan muatan dalam tonase
constraint_2 = rep(0,n_pecah)
# constraint 3
# armada yang mengantar tidak boleh melebihi max armada yang memungkinkan
constraint_3 = rep(0,n_pecah)
# constraint 4
# rute yang dilalui tidak melebihi max rute
constraint_4 = rep(0,n_pecah)

# proses menghitung semua constraint
for(i in 1:n_pecah){
  temp_1 = pecah[[i]]
  # constraint 1
  c_1              = sum(temp_1$order_kubikasi) - mean(temp_1$max_cap_kubikasi)
  c_1              = max(c_1,0)
  constraint_1[i]  = beta * c_1^2

  # constraint 2
  c_2              = sum(temp_1$order_tonase) - mean(temp_1$max_cap_tonase)
  c_2              = max(c_2,0)
  constraint_2[i]  = beta * c_2^2

  # constraint 3
  c_3              = mean(temp_1$armada) - mean(temp_1$max_armada)
  c_3              = max(c_3,0)
  constraint_3[i]  = beta * c_3^2

  # constraint 4
  c_4              = nrow(temp_1) - mean(temp_1$max_titik)
  c_4              = max(c_4,0)
  constraint_4[i]  = beta * c_4^2

}

# ada beberapa constraint yang hanya bisa dilihat per tanggal kirim
pecah      = df_temp_3 %>% group_split(tanggal_kirim)
n_pecah    = length(pecah)
```

```
  # constraint 5
  # total waktu loading
  constraint_5 = rep(0,n_pecah)

  # proses menghitung semua constraint
  for(i in 1:n_pecah){
    temp_1 = pecah[[i]]
    # constraint 5
    c_5              = sum(temp_1$loading_time) - df_gudang
    c_5              = max(c_5,0)
    constraint_5[i]  = beta * c_5^2
  }

  output = jarak_total + sum(constraint_1) + sum(constraint_2) +
           sum(constraint_3) + sum(constraint_4) + sum(constraint_5)

  return(output)
}


# ================================================================================
# fungsi untuk rotasi dan kontraksi
ro_kon_1 = function(list,center){
  Xt_1 = list
  # kita rotasikan dan konstraksikan
  X1 = mat_rotasi %*% (Xt_1 - center_1)
  X1 = center_1 + (.7 * X1)
  X1 = ifelse(X1 <= 1,1,X1)
  X1 = ifelse(X1 >= n_armada,n_armada,X1)
  return(X1)
}

# fungsi untuk rotasi dan kontraksi
ro_kon_2 = function(list,center){
  Xt_2 = list
  # kita rotasikan dan konstraksikan
  X2 = mat_rotasi %*% (Xt_2 - center_2)
  X2 = center_2 + (.7 * X2)
  X2 = ifelse(X2 <= 1,1,X2)
  X2 = ifelse(X2 >= 7,7,X2)
  return(X2)
}
```

```
# ============================================================================
# sekarang kita akan mulai bagian SDOA
n_toko   = nrow(df_toko)
n_armada = nrow(df_jenis_armada)
n_solusi = 3000
n_sdoa   = 50

# kita akan gunakan prinsip parallel processing
# paralel
library(parallel)
numCores = 5 # banyaknya cores yang digunakan

# list pertama yakni armada
# bikin dummy
df_dummy = data.frame(id = 1:n_solusi,
                      n_toko,
                      n_armada)
hasil = mcmapply(armada_generate,df_dummy$n_toko,df_dummy$n_armada,
                 mc.cores = numCores)
# pecah ke list
solusi_1 = lapply(seq_len(ncol(hasil)), function(i) hasil[,i])


# list pertama yakni tanggal
solusi_2 = vector("list",n_solusi)

# kita generate calon solusi terlebih dahulu
for(i in 1:n_solusi){
  solusi_2[[i]] = tanggal_generate(n_toko,df_order)
  print(i)
}

# buat matriks rotasi
mat_rotasi = buat_rot_mat(2*pi / 30,n_toko)

# initial condition
f_hit = c()

# kita hitung dulu initial function objective
f_hit = mcmapply(obj_func,solusi_1,solusi_2,mc.cores = numCores)


# kita mulai perhitungannya di sini
for(iter in 1:n_sdoa){
  # kita cari dulu mana yang akan jadi pusat
```

```
  n_bhole = which.min(f_hit)

  # kita jadikan center of gravity
  center_1 = solusi_1[[n_bhole]]
  center_2 = solusi_2[[n_bhole]]

  solusi_1_new = mcmapply(ro_kon_1,solusi_1,center_1)
  solusi_1     = lapply(seq_len(ncol(solusi_1_new)), function(i) solusi_1_new[,i])

  solusi_2_new = mcmapply(ro_kon_2,solusi_2,center_2)
  solusi_2     = lapply(seq_len(ncol(solusi_2_new)), function(i) solusi_2_new[,i])

  # kita hitung kembali function objective
  f_hit = mcmapply(obj_func,solusi_1,solusi_2,mc.cores = numCores)

  pesan = paste0("Iterasi ke: ",iter," hasilnya: ",min(f_hit))
  print(pesan)
}


# kita akan cek solusinya
n_bhole = which.min(f_hit)

# solusinya
center_1 = solusi_1[[n_bhole]]
center_2 = solusi_2[[n_bhole]]

# kita buat dulu ke data frame untuk mengecek semua informasi yang ada
df_temp_1 = df_toko %>% select(nama_toko,long,lat,max_armada)
df_temp_2 = df_order %>% select(nama_toko,order_kubikasi,order_tonase)
df_temp_3 = merge(df_temp_1,df_temp_2) %>% distinct()
df_temp_3$id_armada     = round(as.vector(center_1),0) # kita rounding dulu ya
df_temp_3$tanggal_kirim = round(as.vector(center_2),0) # kita rounding dulu ya
df_temp_3 = merge(df_temp_3,df_jenis_armada)


save(df_temp_3,file = "cibitung done.rda")
```