Penelitian Mandiri Sains Komputasi I

Update Progress

Mohammad Rizka Fadhli
Ikang
20921004@mahasiswa.itb.ac.id

20 November 2021

Contents

List of Figures

List of Tables

1 SEJARAH

1.1 Optimisasi

Optimisasi adalah **proses mencari nilai yang optimal** dari suatu masalah tertentu. Dalam matematika, optimisasi merujuk pada pencarian nilai minimal atau maksimal dari suatu *fungsi real*[1]. Notasi matematikanya dapat ditulis sebagai berikut:

Misalkan suatu fungsi f yang memetakan dari himpunan A ke bilangan *real*.

$$f : A \to \mathbb{R}$$

Cari suatu nilai $x_0 \in A$ sedemikian sehingga:

- $f(x_0) \leq f(x), \forall x \in A$ untuk proses **minimalisasi**.
- $f(x_0) \geq f(x), \forall x \in A$ untuk proses **maksimalisasi**.

Di dalam kalkulus, kita mengetahui salah satu pendekatan optimisasi di fungsi satu variabel bisa didapatkan dari turunan pertama yang bernilai **nol** (bisa berupa nilai maksimum atau minimum dari fungsi tersebut).

Nilai $x_0 \in [a, b]$ disebut minimum atau maksimum di f unimodal saat memenuhi:

$$\frac{d}{dx} f(x_0) = 0$$

Pierre De Fermat dan **Joseph-Louis Lagrange** adalah orang-orang yang pertama kali menemukan formula kalkulus untuk mencari nilai optimal. Sementara **Isaac Newton** dan **Johann C. F.Gauss** mengusulkan metode iteratif untuk mencari nilai optimal[2].

Salah satu bentuk optimisasi yakni *linear programming* dimulai oleh **Leonid Kantorovich** pada 1939. **Metode Simplex** merupakan salah satu metode penyelesaian optimisasi yang

[1] https://id.wikipedia.org/wiki/Optimisasi
[2] https://empowerops.com/en/blogs/2018/12/6/brief-history-of-optimization

terkenal, pertama kali diperkenalkan pada 1947 oleh **George Dantzig** sementara di tahun yang sama *Theory of Duality* diperkenalkan oleh **John von Neumann**.

1.2 Riset Operasi

Riset operasi adalah metode antar disiplin ilmu yang digunakan untuk menganalisa masalah nyata dan membuat keputusan untuk kegiatan operasional organisasi atau perusahaan[3].

Riset operasi dimulai pada era Perang Dunia II. Oleh karena peperangan, diperlukan suatu cara yang efektif untuk mengalokasikan *resources* yang ada sehingga pihak militer Inggris dan Amerika Serikat mengumpulkan ilmuwan-ilmuwan untuk mencari pendekatan yang saintifik dalam memecahkan masalah(Hillier and Lieberman 2001).

Pada tahun 1940, sekelompok *researchers* yang dipimpin oleh **PMS Blackett** dari ***the University of Manchester*** melakukan studi tentang **Sistem Radar Baru Anti Pesawat Terbang**. Kelompok *researchers* ini sering dijuluki sebagai **Kelompok Sirkus Blackett** (*Blackett's circus*). Julukan ini terjadi karena keberagaman latar belakang disiplin ilmu para *researchers* tersebut. Mereka terdiri dari disiplin ilmu fisiologi, matematika, astronomi, tentara, surveyor, dan fisika. Pada 1941, kelompok ini terlibat dalam penelitian radar deteksi kapal selam dan pesawat terbang. *Blackett* kemudian memimpin *Naval Operational Research* pada Angkatan Laut Kerajaan Inggris Raya. Prinsip-prinsip ilmiah yang digunakan untuk mengambil keputusan dalam suatu operasi dinamai sebagai **Riset Operasi**(Parmono 2007).

Saat Amerika Serikat mulai terlibat pada Perang Dunia II, prinsip riset operasi juga digunakan untuk berbagai operasi militer mereka. Kelompok riset operasi AS bertugas untuk menganalisis serangan udara dan laut tentara NAZI Jerman.

Selepas Perang Dunia II, penerapan riset operasi dinilai bisa diperluas ke dunia ekonomi, bisnis,*engineering*, dan sosial. Riset operasi banyak berkaitan dengan berbagai disiplin ilmu seperti matematika, statistika, *computer science*, dan lainnya. Tidak jarang beberapa pihak menganggap riset operasi itu *overlapping* dengan disiplin-disiplin ilmu tersebut.

Oleh karena tujuan utama dari aplikasi riset operasi adalah tercapainya **hasil yang optimal**

[3] Pengantar Riset Operasi dan Optimisasi, KampusX: PO101

dari semua kemungkinan perencanaan yang dibuat. Maka **pemodelan matematika dan optimisasi** bisa dikatakan sebagai disiplin utama dari riset operasi.

2 OPTIMISASI

2.1 Bahasan dalam Optimisasi

Bahasan dalam optimisasi dapat dikategorikan menjadi:

- Pemodelan masalah nyata menjadi masalah optimisasi.
- Pembahasan karakteristik dari masalah optimisasi dan keberadaan solusi dari masalah optimisasi tersebut.
- Pengembangan dan penggunaan algoritma serta analisis numerik untuk mencari solusi dari masalah tersebut.

2.2 Masalah Optimisasi

Masalah optimisasi adalah masalah matematika yang mewakili masalah nyata (*real*). Dari ekspresi matematika tersebut, ada beberapa hal yang perlu diketahui[4], yakni:

1. **Variabel** adalah suatu simbol yang memiliki banyak nilai dan nilainya ingin kita ketahui. Setiap nilai yang mungkin dari suatu variabel muncul akibat suatu kondisi tertentu di sistem.
2. **Parameter** di suatu model matematika adalah suatu konstanta yang menggambarkan suatu karakteristik dari sistem yang sedang diteliti. Parameter bersifat *fixed* atau *given*.
3. ***Constraints*** (atau kendala) adalah kondisi atau batasan yang harus dipenuhi. Kendala-kendala ini dapat dituliskan menjadi suatu persamaan atau pertaksamaan. Suatu masalah optimisasi dapat memiliki hanya satu kendala atau banyak kendala.

[4] Pengantar Riset Operasi dan Optimisasi, KampusX: PO101

4. ***Objective function*** adalah satu fungsi (pemetaan dari variabel-varibel keputusan ke suatu nilai di daerah *feasible*) yang nilainya akan kita minimumkan atau kita maksimumkan.

Ekspresi matematika dari model optimisasi adalah sebagai berikut:

Cari x yang meminimumkan $f(x)$ dengan kendala $g(x) = 0, h(x) \leq 0$ dan $x \in D$.

Dari ekspresi tersebut, kita bisa membagi-bagi masalah optimisasi tergantung dari:

1. Tipe variabel yang terlibat.
2. Jenis fungsi yang ada (baik *objective function* ataupun *constraints*).

2.3 Jenis-Jenis Masalah Optimisasi

Masalah optimisasi bisa dibagi dua menjadi dua kategori berdasarkan tipe *variables* yang terlibat[5], yakni:

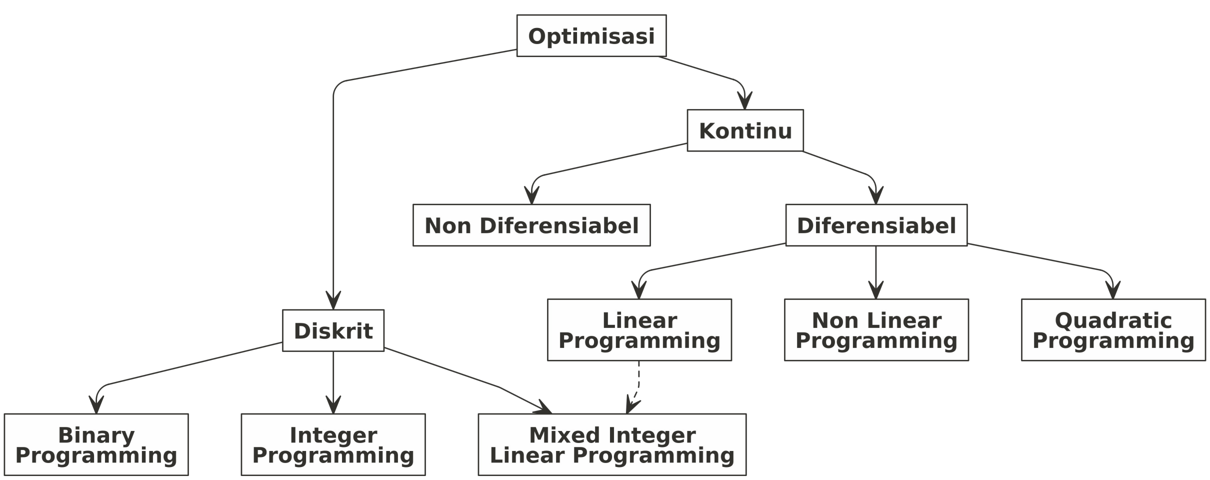

Figure 1: Optimisasi Berdasarkan Jenis Variabel

[5] Optimization problem. https://en.wikipedia.org/wiki/Optimization_problem

1. *Discrete Optimization*: merupakan masalah optimisasi di mana variabel yang terkait merupakan variabel diskrit, seperti *binary* atau *integer* (bilangan bulat). Namun pada masalah optimisasi berbentuk *mixed integer linear programming*, dimungkinkan suatu masalah optimisasi memiliki berbagai jeni variabel yang terlibat (integer dan kontinu sekaligus).
2. *Continuous Optimization*: merupakan masalah optimisasi di mana variabel yang terkait merupakan variabel kontinu (bilangan *real*). Pada masalah optimisasi jenis ini, fungsi-fungsi yang terlibat bisa diferensiabel atau tidak. Konsekuensinya adalah pada metode penyelesaiannya.

Selain itu, kita juga bisa membagi masalah optimisasi berdasarkan **kepastian nilai *variable* dan parameter** yang dihadapi sebagai berikut:

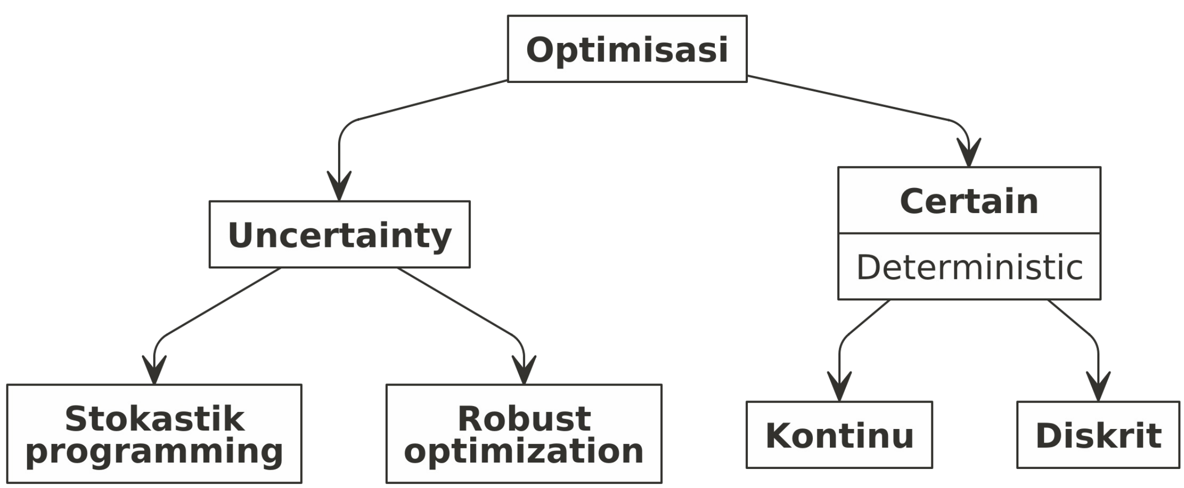

Figure 2: Optimisasi Berdasarkan Kepastian Nilai

1. *Optimization under uncertainty*[6]; Pada beberapa kasus di dunia *real*, data dari masalah tidak dapat diketahui secara akurat karena berbagai alasan. Hal ini mungkin terjadi akibat:

   - Kesalahan dalam pengukuran, atau

[6] https://neos-guide.org/content/optimization-under-uncertainty

- Data melibatkan sesuatu di masa depan yang belum terjadi atau tidak pasti. Contoh: *demand* produk, harga barang, dan sebagainya.

2. *Deterministic optimization*;
    - Model deterministik adalah model matematika di mana nilai dari semua parameter dan variabel yang terkandung di dalam model merupakan satu nilai pasti[7].
    - Pendekatan deterministik memanfaatkan sifat analitik masalah untuk menghasilkan barisan titik yang konvergen ke solusi optimal(Lin, Tsai, and Yu 2012).
    - Semua algoritma perhitungan mengikuti pendekatan matematis yang ketat(Cavazzuti 2013).

2.4 *Supplier Selection Problem*

Tema penelitian yang diangkat adalah *supplier selection problem.*

Latar Belakang Suatu perusahaan makanan dan minuman memproduksi > 130 jenis produk perbulannya dengan kuantitas yang disesuaikan dengan *sales forecast* pada bulan tersebut. Untuk memproduksi satu jenis produk, dibutuhkan beberapa *raw material.* Perusahaan tersebut menggunakan strategi *multi supplier* untuk menjaga keberadaan stok *raw material.* Artinya satu jenis *raw material* bisa dibeli di beberapa *supplier* yang berbeda. Akibatnya timbul perbedaan dalam hal:

1. Harga (per ton),
2. Minimum tonase pembelian,
3. *Delivery time.*

Mengenai kualitas *raw material*, **sudah dipastikan** bahwa tidak ada perbedaan antar *supplier* karena pemilihannya melalui proses seleksi yang ketat.

[7] Pengantar Riset Operasi dan Optimisasi, KampusX: PO101

Masalah yang Dihadapi adalah meminimalkan ongkos pembelian *raw material* dengan tetap mengamankan stok *raw material* agar proses produksi tetap berjalan.

Masalah ini termasuk ke dalam masalah optimisasi deterministik yakni *mixed integer linear programming* (*MILP*), alasannya:

1. Parameter dan variabel yang terlibat merupakan suatu nilai pasti.
2. Variabel yang terlibat meliputi:
   - *Binary* karena melibatkan pengambilan keputusan *raw material* dari *supplier* mana yang harus dipesan.
   - *Continuous* karena melibatkan angka kuantitas *raw material* yang harus dipesan.
3. Fungsi *objective* dan *constraints* masih berupa *linear*.
   - Meminimumkan ongkos pembelian $harga \times tonase$.
   - Kapasitas gudang *raw material* dan rencana produksi produk.
   - dan seterusnya.

3 JENIS OPTIMISASI

3.1 *Linear Programming*

Linear programming adalah bentuk metode optimisasi sederhana yang memanfaatkan relasi linear (semua fungsi dan *constraints* merupakan fungsi linear).

3.1.1 Contoh Masalah *Linear Programming*

Saya memiliki area parkir seluas 1.960 m^2. Luas rata-rata untuk mobil berukuran kecil adalah 4 m^2 dan mobil besar adalah 20 m^2. Daya tampung maksimum hanya 250 kendaraan, biaya parkir mobil kecil adalah Rp 7.000 per jam dan mobil besar adalah Rp 12.000 per jam. Jika dalam 1 jam area parkir saya terisi penuh dan tidak ada kendaraan yang pergi dan datang, maka berapa pendapatan maksimum yang bisa saya dapatkan dari tempat parkir itu?

Dari kasus di atas kita bisa tuliskan model matematikanya sebagai berikut:

Misal x_1 adalah mobil kecil dan x_2 adalah mobil besar.

$$max(7000x_1 + 12000x_2)$$

Dengan *constraints*:

$$4x_1 + 20x_2 \leq 1960$$

dan

$$x_1 + x_2 \leq 250$$

serta $x_1 \geq 0, x_2 \geq 0$.

3.2 *Integer Programming*

Integer programming adalah bentuk metode optimisasi di mana variabel yang terlibat merupakan bilangan bulat (*integer*). Jika fungsi-fungsi yang terkait merupakan *linear*, maka disebut dengan *integer linear programming*.

Sebagai contoh, variabel yang merupakan bilangan bulat adalah banyak orang.

3.2.1 Contoh *Integer Programming*

Jadwal Kebutuhan Tenaga Kesehatan Suatu rumah sakit membutuhkan tenaga kesehatan setiap harinya dengan spesifikasi berikut:

Table 1: Tabel Kebutuhan Nakes Harian

hari	Min Nakes Required	Max Nakes Required
Senin	24	29
Selasa	22	27
Rabu	23	28
Kamis	11	16
Jumat	16	21
Sabtu	20	25
Minggu	12	17

Di rumah sakit tersebut berlaku kondisi sebagai berikut:

1. Setiap nakes hanya diperbolehkan bekerja selama 5 hari berturut-turut dan harus libur selama 2 hari berturut-turut.
2. Tidak ada pemberlakuan *shift* bagi nakes.

Berapa banyak nakes yang harus dipekerjakan oleh rumah sakit tersebut? Bagaimana konfigurasi penjadwalannya?

Untuk memudahkan dalam mencari solusi permasalahan di atas, kita bisa membuat tabel ilustrasi berikut:

Table 2: Konfigurasi Penjadwalan Nakes

hari	Min Nakes Required	Max Nakes Required	x1	x2	x3	x4	x5	x6	x7
Senin	24	29	x			x	x	x	x
Selasa	22	27	x	x			x	x	x
Rabu	23	28	x	x	x			x	x
Kamis	11	16	x	x	x	x			x
Jumat	16	21	x	x	x	x	x		
Sabtu	20	25		x	x	x	x	x	
Minggu	12	17			x	x	x	x	x

Kolom $x_i, i = 1, 2, 3, 4, 5, 6, 7$ menandakan kelompok nakes yang perlu dipekerjaan pada hari-hari tertentu. Setiap nilai x_i tersebut merupakan **bilangan bulat positif** $x \geq 0, x \in \mathbb{Z}$.

Dari ilustrasi di atas, kita bisa membuat model optimisasinya sebagai berikut:

Objective Function

$$\min \sum_{i=1}^{7} x_i$$

Constraints

- Hari Senin: $24 \leq \sum x_i \leq 29, i \in \{1, 4, 5, 6, 7\}$.
- Hari Selasa: $22 \leq \sum x_i \leq 27, i \in \{1, 2, 5, 6, 7\}$.
- Hari Rabu: $23 \leq \sum x_i \leq 28, i \in \{1, 2, 3, 6, 7\}$.
- Hari Kamis: $11 \leq \sum x_i \leq 16, i \in \{1, 2, 3, 4, 7\}$.
- Hari Jumat: $16 \leq \sum x_i \leq 21, i \in \{1, 2, 3, 4, 5\}$.
- Hari Sabtu: $20 \leq \sum x_i \leq 25, i \in \{2, 3, 4, 5, 6\}$.
- Hari Minggu: $12 \leq \sum x_i \leq 17, i \in \{3, 4, 5, 6, 7\}$.

Kita juga perlu perhatikan bahwa $x_i \geq 0, i \in \{1, 2, 3, 4, 5, 6, 7\}$.

3.3 *Binary Programming*

Binary programming adalah bentuk metode optimisasi di mana variabel yang terlibat merupakan bilangan biner (0,1). Biasanya metode ini dipakai dalam masalah penjadwalan yang memerlukan prinsip *matching* antar kondisi yang ada.

3.3.1 Contoh *Binary Programming*

Jadwal Tatap Muka Terbatas Sekolah Beberapa minggu ke belakang, kasus harian Covid semakin menurun. Pemerintah mulai melonggarkan aturan PPKM yang mengakibatkan sekolah-sekolah mulai menggelar pengajaran tatap muka terbatas (PTMT) untuk siswanya secara *offline*.

Suatu sekolah memiliki kelas berisi 20 orang siswa. Mereka hendak menggelar PTMT dengana aturan sebagai berikut:

1. PTMT digelar dari Senin hingga Jumat (5 hari).
2. Dalam sehari, siswa yang boleh hadir dibatasi 4-8 orang saja.
3. Dalam seminggu, diharapkan siswa bisa hadir 2-3 kali.
4. Siswa yang hadir di selang sehari baru bisa hadir kembali.

Dari uraian di atas, kita bisa membuat model optimisasinya sebagai berikut:

Saya definisikan $x_{i,j} \in (0, 1)$ sebagai bilangan biner di mana $i \in \{1, 2, .., 20\}$ menandakan siswa dan $j \in \{1, 2, .., 5\}$ menandakan hari. Berlaku:

$$x_{i,j} = \begin{cases} 0, \text{ siswa i tidak masuk di hari j} \\ 1, \text{ siswa i masuk di hari j} \end{cases}$$

Objective Function

Tujuan utama kita adalah memaksimalkan siswa yang hadir.

$$\max \sum_{j=1}^{5} \sum_{i=1}^{20} x_{i,j}$$

Constraints

Dalam sehari, ada pembatasan jumlah siswa yang hadir.

$$4 \leq \sum_{i} x_{i,j} \leq 8, j \in \{1, 2, .., 5\}$$

Dalam seminggu, siswa hadir dalam frekuensi tertentu.

$$2 \leq \sum_{j} x_{i,j} \leq 3, i \in \{1, 2, .., 20\}$$

Ada jeda sehari agar siswa bisa masuk kembali.

$$x_{i,j} + x_{i,j+1} \leq 1$$

Jangan lupa bahwa $x_{i,j} \geq 0$.

3.4 *Mixed Integer Linear Programming*

Pada bagian sebelumnya, kita telah membahas masalah optimisasi dengan variabel berupa diskrit dan kontinu. Permasalahan *real* yang ada di kehidupan sehari-hari biasanya merupakan memiliki variabel yang *mixed* antara keduanya. Oleh karena itu, ada metode yang disebut dengan *mixed integer linear programming*. Pada masalah optimisasi tipe ini, *decision variables* yang terlibat bisa saja berupa *binary*, *integer*, dan *continuous* sekaligus.

3.4.1 Menyelesaikan *MILP*

MILP secara eksak bisa diselesaikan dengan metode *simplex* dengan dikombinasikan dengan teknik *branch and bound*. Penjelasan terkait ini akan dibahas pada bab 6.

3.4.2 Contoh *MILP*

Pemilihan dan Penentuan Item Produksi Suatu pabrik makanan dan minuman berencana untuk membuat tiga produk baru yang bisa diproduksi di dua *plants* yang berbeda.

Table 3: Tabel Runtime Item Produk per Plant (harian - dalam jam)

Produk	Runtime Plant 1	Runtime Plant 2
Item 1	3	4
Item 2	4	6
Item 3	2	2

Plant 1 memiliki maksimum *working hours* sebesar 30 jam perhari.

Plant 2 memiliki maksimum *working hours* sebesar 40 jam perhari.

Table 4: Tabel Profit dan Potensi Sales Item Produk

Produk	Profit per ton	Sales potential per ton
Item 1	5	7
Item 2	7	5
Item 3	3	9

Masalah timbul saat mereka harus memilih **dua dari tiga** produk baru tersebut yang harus di produksi. Selain itu, mereka juga harus memilih **satu dari dua** *plants* yang memproduksi *items* tersebut.

Misalkan saya definisikan:

- $x_i \geq 0, i = 1, 2, 3$ sebagai `berapa ton` yang harus diproduksi dari item i.
- $y_i \in [0, 1], i = 1, 2, 3$ sebagai *binary*.

  - Jika bernilai 0, maka produk i tidak dipilih.
  - Jika bernilai 1, maka produk i dipilih.

- $z \in [0,1]$ sebagai *binary*.
  - Jika bernilai 0, maka *plant* pertama dipilih.
  - Jika bernilai 1, maka *plant* kedua dipilih.

Saya akan mendefinisikan suatu variabel *dummy* $M = 99999$ berisi suatu nilai yang besar. Kelak variabel ini akan berguna untuk *reinforce model* (metode pemberian *penalty*) agar bisa memilih *items* dan *plants* secara bersamaan.

Objective function dari masalah ini adalah memaksimalkan *profit*.

$$\max \sum_{i=1}^{3} x_i \times \text{profit}_i$$

Constraints dari masalah ini adalah:

Tonase produksi tidak boleh melebihi angka *sales potential* per items.

$$x_i \leq \text{sales potential}_i, i = 1, 2, 3$$

Kita akan memilih dua produk sekaligus menghitung tonase. Jika produk tersebut **dipilih**, maka akan ada angka tonase produksinya. Sebaliknya, jika produk tersebut **tidak dipilih**, maka tidak ada angka tonase produksinya.

$$x_i - y_i \times M \leq 0, i = 1, 2, 3$$

$$\sum_{i=1}^{3} y_i \leq 2$$

Kita akan memilih *plant* dari waktu produksinya.

$$3x_1 + 4x_2 + 2x_3 - M \times z \leq 30$$

$$4x_1 + 6x_2 + 2x_3 + M \times z \leq 40 + M$$

4 ALGORITMA PENYELESAIAN OPTIMISASI

Pada bagian ini kita akan membahas macam-macam algoritma yang digunakan untuk menyelesaikan masalah optimisasi.

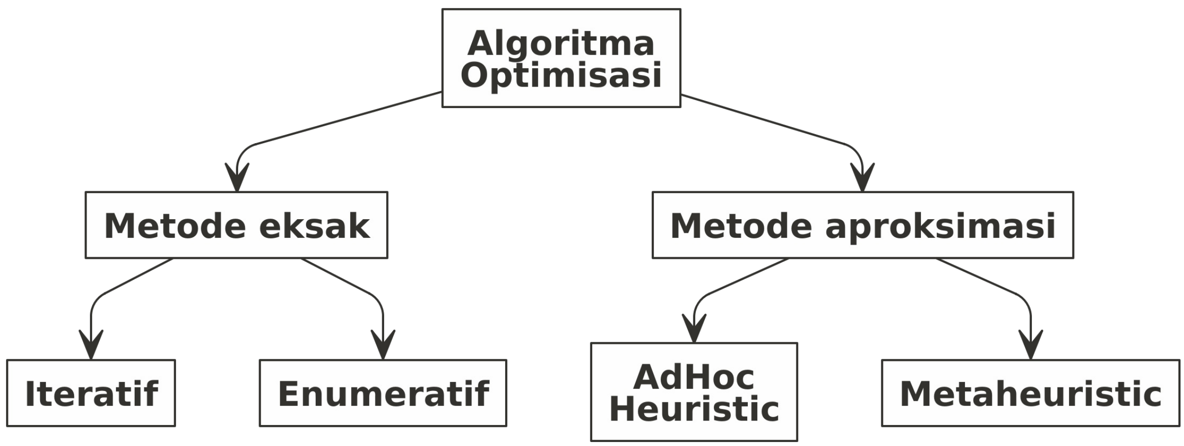

Figure 3: Algoritma Penyelesaian Optimisasi

Secara garis besar ada dua kelompok besar algoritma optimisasi, yakni:

1. *Exact method*,
2. *Approximate method*.

Perbedaan keduanya adalah pada **konsep atau pendekatan apa yang digunakan** untuk menyelesaikan masalah optimisasi. Kita akan bahas satu-persatu pada bagian selanjutnya.

Dalam beberapa kasus, kita bisa mendapatkan *exact method* bisa untuk menyelesaikan masalah optimisasi dengan efisien. Namun di kasus lain yang lebih kompleks tidak demikian. Kelemahan utama metode *exact* adalah pada waktu komputasinya yang relatif lebih lama.

4.1 *Exact Method*

Ciri khas dari *exact method* adalah metode ini menjamin penyelesaian yang optimal karena menggunakan pendekatan analitis (Rothlauf 2011). Salah satu contoh metode eksak adalah *Simplex Method*.

4.2 *Approximate Method*

Ciri khas dari *approximate method* adalah metode ini tidak menjamin penyelesaian yang optimal karena bersifat *aproksimasi* atau pendekatan atau hampiran(Geovanni and Summa 2018). Oleh karena itu kita perlu melakukan definisi di awal **seberapa dekat** nilai **hampiran** tersebut bisa kita terima.

Metode ini bisa dibagi menjadi dua berdasarkan keterkaitannya dengan suatu masalah, yakni:

1. *Heuristic*, metode ini bersifat *problem dependent*. Artinya metode tersebut hanya bisa dipakai untuk jenis permasalahan tertentu.
   - Contoh: metode *nearest neighborhood* hanya bisa dipakai untuk menyelesaikan masalah dalam lingkup *travelling salesperson problem* (**TSP**).
2. *Meta heuristic*, metode ini bersifat *problem independent*. Artinya metode tersebut tidak tergantung dari jenis permasalahan tertentu. Contoh:
   - *Genetic algorithm*.
   - *Simulated annealing*.
   - *Spiral optimization* untuk menyelesaikan masalah *mixed integer non linear programming* (Kania and Sidarto 2016).
   - *Artifical bee colony algorithm*.

Namun demikian kedua metode ini bisa saling melengkapi dalam prakteknya.

5 METODE *SIMPLEX*

Metode *simplex* adalah salah satu metode yang paling umum digunakan dalam menyelesaikan permasalahan *linear programming*. Metode ini dikembangkan oleh seorang profesor matematika bernama George Dantzig[8] pada 1947 pasca perang dunia II. Sedangkan nama *simplex* diusulkan oleh Theodore Motzkin[9].

Metode *simplex* menggunakan prosedur aljabar(Hillier and Lieberman 2001). Namun *underlying concept* dari metode ini adalah *geometric*.

5.1 Metode Simplex dengan Ilustrasi Geometris

Jika kita bisa memahami konsep geometrinya, kita bisa mengetahui bagaimana cara kerjanya dan kenapa metode ini sangat efisien.

Saya akan ambil satu contoh masalah optimisasi sederhana untuk memberikan ilustrasi bagaimana cara kerja metode ini.

Contoh Masalah Optimisasi Cari $x_1, x2$ yang $\max\left(Z = 3x_1 + 5x_2\right)$ dengan *constraints*:

$$x1 \leq 4$$
$$2x_2 \leq 12$$
$$3x_1 + 2x_2 \leq 18$$
$$\text{serta } x_1 \geq 0, x_2 \geq 0$$

Masalah di atas jika dibuat grafiknya:

[8] https://en.wikipedia.org/wiki/George_Dantzig
[9] https://en.wikipedia.org/wiki/Theodore_Motzkin

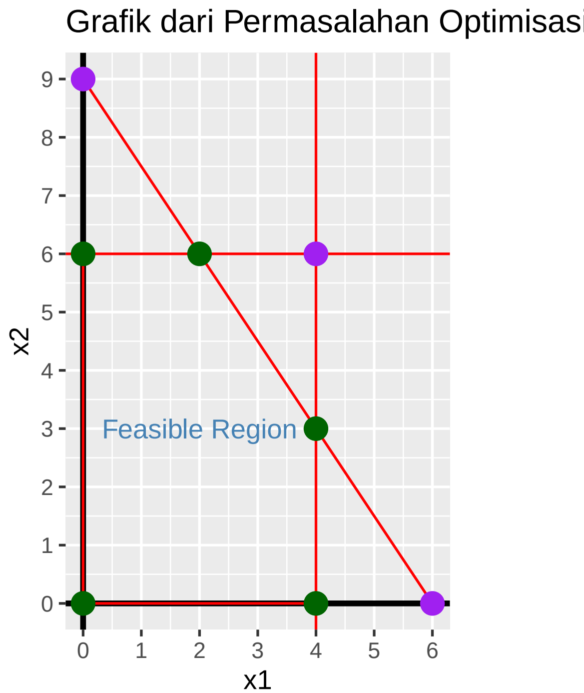

Figure 4: Grafik Permasalahan Optimisasi

Titik-titik hijau merupakan **beberapa titik** solusi yang *feasible* karena berada pada area penerimaan seluruh *constraints* yang ada. Titik hijau ini menjadi spesial karena berada pada perpotongan 2 garis *constraints*. Selanjutnya titik hijau ini akan didefinisikan sebagai **CPF** (*corner point feasible*).

> *For a linear programming problem with n decision variables, each of its corner-point solutions lies at the intersection of n constraint boundaries.* (Hillier and Lieberman 2001)

Sedangkan titik ungu merupakan titik solusi non *feasible* karena solusi yang ada tidak berlaku untuk semua *constraints*.

Table 5: Titik yang termasuk ke dalam CPF

Titik.ke	CPF
1	(0, 0)

Titik.ke	CPF
2	(0, 6)
3	(2, 6)
4	(4, 3)
5	(4, 0)

Properties of CPF Solutions Untuk setiap permasalahan *linear programming* yang memiliki *feasible solutions* dan *feasible region* yang terbatas, berlaku:

- **Property 1**:
  - (a) If there is exactly one optimal solution, then it must be a **CPF solution**.
  - (b) If there are multiple optimal solutions (and a bounded feasible region), then at least two must be adjacent CPF solutions.
- **Property 2**: There are only a **finite number** of CPF solutions.
- **Property 3**: If a CPF solution has no adjacent CPF solutions that are better (as measured by Z), then there are no better CPF solutions anywhere. Therefore, **such a CPF solution is guaranteed to be an optimal solution** (by Property 1), assuming only that the problem possesses at least one optimal solution (guaranteed if the problem possesses feasible solutions and a bounded feasible region).

Properties di atas menjamin keberadaan solusi optimal pada CPF dari suatu masalah optimisasi *linear programming*.

Untuk mulai melakukan metode simplex kita perhatikan kembali grafik di atas. Kita bisa temukan beberapa pasang **CPF** berbagi *constraint* yang sama satu sama lain.

Sebagai contoh:

1. CPF_1 dan CPF_2 berbagi *constraint* yang sama, yakni saat $x_1 \geq 0$.
2. CPF_2 dan CPF_3 berbagi *constraint* yang sama, yakni saat $x_2 \leq 6$.

Definisi umum:

> *For any linear programming problem with n decision variables, two CPF solutions are **adjacent** to each other if they share $n-1$ constraint boundaries. The two adjacent CPF solutions are connected by a line segment that lies on these same shared constraint boundaries. Such a line segment is referred to as an **edge** of the feasible region.*

Feasible region di atas memiliki 5 *edges* di mana setiap 2 *edges* memotong / memunculkan **CPF**. Setiap **CPF** memiliki 2 **CPF** lainnya yang *adjacent.*

Table 6: Adjacent CPF

Titik.ke	CPF	Adjacent.CPF
1	(0, 0)	(0, 6) dan (4, 0)
2	(0, 6)	(2, 6) dan (0, 0)
3	(2, 6)	(4, 3) dan (0, 6)
4	(4, 3)	(4, 0) dan (2, 6)
5	(4, 0)	(0, 0) dan (4, 3)

CPF pada kolom pertama *adjacent* terhadap dua **CPF** di kolom setelahnya tapi kedua **CPF** tersebut tidak saling *adjacent* satu sama lain.

> ***Optimality test:*** *Consider any linear programming problem that possesses at least one optimal solution. If a CPF solution has no adjacent **CPF** solutions that are better (as measured by Z), then it must be an optimal solution.*

Berdasarkan *optimality test* tersebut, kita bisa mencari solusi optimal dari **CPF** dengan cara mengambil **initial CPF** untuk dites secara rekursif.

- **STEP 1** Pilih *initial* **CPF**, misal $(0, 0)$. Kita akan hitung nilai $Z(0, 0) = 0$. Bandingkan dengan *adjacent* **CPF**-nya, yakni $Z(0, 6) = 30$ dan $Z(4, 0) = 12$.
- **STEP 2** Oleh karena $Z(0, 6)$ memiliki nilai tertinggi, maka kita akan pilih titik ini di iterasi pertama. Kita akan bandingkan terhadap *adjacent* **CPF**-nya, yakni: $Z(2, 6) = 36$. Perhatikan bahwa *adjacent* **CPF** $(0, 0)$ sudah kita evaluasi pada langkah sebelumnya.
- **STEP 3** Oleh karena $Z(2, 6)$ memiliki nilai tertinggi, maka kita akan pilih titik ini di iterasi kedua. Kita akan bandingkan terhadap *adjacent* **CPF**-nya, yakni: $Z(4, 3) = 27$. Kita dapatkan bahwa titik $(2, 6)$ menghasilkan Z tertinggi.

Kesimpulan: $(2, 6)$ merupakan titik yang bisa memaksimumkan Z.

5.1.1 ***Flowchart*** Metode Simplex dari Contoh Masalah

Secara garis besar, *flowchart* dari metode simplex untuk masalah di atas adalah:

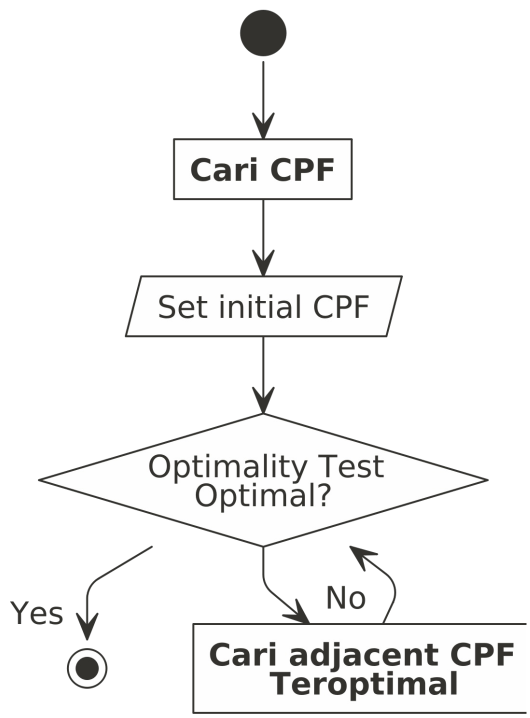

Figure 5: Algoritma Metode Simplex

Algoritma di atas akan sangat mudah dilakukan saat kita berhadapan dengan masalah optimisasi dengan 2 *decision variables* (atau 3 *decision variables*). Pada contoh di atas ada x_1, x_2.

Bagaimana jika masalah yang dihadapi memiliki banyak *decision variables*?

Tentunya kita tidak bisa melakukan analisa secara visual seperti di atas. Namun kita bisa menggunakan bantuan aljabar dan operasi baris elementer untuk menemukan solusi yang optimal.

5.2 Transisi Geometris ke Aljabar

Pada penjelasan sebelumnya kita bisa melihat ilustrasi geometris dari suatu masalah optimisasi di mana solusi berada di **CPF**. Namun jika kita berhadapan dengan $n > 2$ variabel, kita tidak bisa menggambarkan visualnya. Oleh karena itu kita akan menggunakan skema aljabar untuk menyelesaikannya.

Ide dasarnya adalah dengan mengubah pertaksamaan yang ada di *constraints* menjadi sebuah persamaan dengan menambahkan beberapa variabel *dummy*. Persamaan-persamaan tersebut akan kita jadikan SPL dan dicari solusinya dengan kondisi **semua kombinasi di mana** $n - m$ **variabel dibuat sama dengan nol** (m banyaknya persamaan dan n banyaknya variabel).

- $n - m$ variabel yang dibuat **nol** disebut dengan *non basic variables*,
- Sedangkan variabel m sisanya disebut dengan *basic variables*. Solusi dari SPL ini disebut dengan *basic solution* (Taha 2007).

Dengan contoh masalah yang sama dengan sebelumnya, kita akan selesaikan sebagai berikut:

Masalah Optimisasi Cari x_1, x_2 yang $\max\left(Z = 3x_1 + 5x_2\right)$ dengan *constraints*:

$$x1 \leq 4$$
$$2x_2 \leq 12$$
$$3x_1 + 2x_2 \leq 18$$
$$\text{serta } x_1 \geq 0, x_2 \geq 0$$

Pada *constraints* yang mengandung pertaksamaan $\leq$, *right hand side* menunjukkan batas dari *resources* sementara *left hand side* menunjukkan *usage* dari *resources*. Selisih antara *rhs* dan *lhs* menunjukkan **sisa** *resources* yang tidak terpakai. Kita perlu mengubah pertaksamaan yang ada menjadi bentuk persamaan dengan cara menambahkan u, v, w sebagai ***non negative slack variables*** (Taha 2007).

Oleh karena itu kita tuliskan *constraints* menjadi sebagai berikut:

$$x1 + u = 4$$
$$2x_2 + v = 12$$
$$3x_1 + 2x_2 + w = 18$$
$$\text{dengan } x_1 \geq 0, x_2 \geq 0, u \geq 0, v \geq 0, w \geq 0$$

Perhatikan bahwa $m = 3$ dan $n = 5$ sehingga $n - m = 2$. Maka kita akan buat semua kombinasi *non basic variables* (berisi 2 variabel).

```
##  [1] "(x1,x2)" "(x1,u)"  "(x1,v)"  "(x1,w)"  "(x2,u)"  "(x2,v)"  "(x2,w)"
##  [8] "(u,v)"   "(u,w)"   "(v,w)"
```

Kemudian menyelesaikan *SPL* yang ada pada kondisi *non basic variables* tersebut **nol**. Dari masing-masing solusi yang ada, kita akan lihat apakah *feasible* atau tidak? Serta dievaluasi nilai z-nya.

Berikut adalah algoritma dan tabel hasilnya:

```
# set SPL dari constraints
A = data.frame(x1 = c(1,0,3),
               x2 = c(0,2,2),
```

```
               u = c(1,0,0),
               v = c(0,1,0),
               w = c(0,0,1))
# rhs
c = c(4,12,18)
# obj function
obj_f = function(data){
  # filter var
  x1 = sol_val %>% filter(var %in% c("x1"))
  x2 = sol_val %>% filter(var %in% c("x2"))
  # ambil val
  if(nrow(x1) != 1){x1 = 0}else{x1 = x1$value}
  if(nrow(x2) != 1){x2 = 0}else{x2 = x2$value}
  return(3*x1 + 5*x2)
  }

# set template hasil
hasil = data.frame(non_basic_var = paste0("(",non_basic$v1,",",non_basic$v2,")"),
                   basic_var = NA,
                   solusi = NA,
                   z = NA)

# iterasi
for (i in 1:nrow(non_basic)) {
  # siap print basic var
  basic_var = var_[!grepl(non_basic$v1[i],var_)]
  basic_var = basic_var[!grepl(non_basic$v2[i],basic_var)]
  basic_var = paste(basic_var,collapse = ",")
  hasil$basic_var[i] = paste0("(",basic_var,")")
```

```
# hitung solusi SPL
B = A %>% select(-contains(non_basic$v1[i])) %>% select(-contains(non_basic$v2[i])) %>
if(det(B) == 0){sol_print = NA}
else{
  sol = solve(B) %*% c
  sol_print = paste(row.names(sol),sol,sep = " = ")
  sol_print = paste(sol_print,collapse = "; ")
}
hasil$solusi[i] = sol_print

# evaluasi obj function
if(det(B) == 0){z_hit = NA}
else{
  sol_val = data.frame(var = row.names(sol),value = sol)
  z_hit = obj_f(obj_f)
}
hasil$z[i] = z_hit
}
```

Table 7: Hasil Perhitungan Simplex dengan Metode Aljabar

Non Basic Var	Basic Var	solusi	z	feasible
(x1,x2)	(u,v,w)	u = 4; v = 12; w = 18	0	Yes
(x1,u)	(x2,v,w)	NA	NA	No
(x1,v)	(x2,u,w)	x2 = 6; u = 4; w = 6	30	Yes
(x1,w)	(x2,u,v)	x2 = 9; u = 4; v = -6	45	No
(x2,u)	(x1,v,w)	x1 = 4; v = 12; w = 6	12	Yes
(x2,v)	(x1,u,w)	NA	NA	No

Non Basic Var	Basic Var	solusi	z	feasible
(x2,w)	(x1,u,v)	x1 = 6; u = -2; v = 12	18	No
(u,v)	(x1,x2,w)	x1 = 4; x2 = 6; w = -6	42	No
(u,w)	(x1,x2,v)	x1 = 4; x2 = 3; v = 6	27	Yes
(v,w)	(x1,x2,u)	x1 = 2; x2 = 6; u = 2	36	Yes

Terlihat di atas bahwa $\max z = 36$ terletak pada saat $x_1 = 2, x_2 = 6$. Sama persis dengan perhitungan dengan pendekatan geometris.

5.3 Metode Simplex dengan *Tableau*

Pendekatan aljabar di atas bisa kita buat menjadi suatu operasi baris elementer di matriks. Berikut adalah contohnya:

5.3.1 Operasi Baris Elementer Matriks *Simplex*

Cari x, y sehingga $\max(P = 5x + 4y)$ dengan *constraints*:

$$3x + 5y \leq 78$$
$$4x + y \leq 36$$
$$\text{serta } x \geq 0, y \geq 0$$

Pada *constraints* yang mengandung pertaksamaan $\leq$, *right hand side* menunjukkan batas dari *resources* sementara *left hand side* menunjukkan *usage* dari *resources*. Selisih antara *rhs* dan *lhs* menunjukkan **sisa** *resources* yang tidak terpakai.

Kita perlu mengubah pertaksamaan yang ada menjadi bentuk persamaan dengan cara menambahkan u, w sebagai ***non negative slack variables*** (Taha 2007). Fungsi objectif P juga harus diubah (dipindah sisi namun P tetap positif).

$$3x + 5y + u = 78, \text{ dengan } u \geq 0$$
$$4x + y + w = 36, \text{ dengan } w \geq 0$$
$$-5x - 4y + P = 0$$

Setelah itu kita buat matriks (dalam hal ini saya akan buatkan tabelnya) sebagai berikut:

Table 8: Initial Condition Bentuk Matriks Simplex

x	y	u	w	P	b
3	5	1	0	0	78
4	1	0	1	0	36
-5	-4	0	0	1	0

STEP 1 Kita akan pilih kolom yang memiliki nilai **negatif terbesar** pada baris terakhir, yakni kolom x. Selanjutnya kita akan pilih baris mana yang akan menjadi pivot dengan cara menghitung rasio $\frac{b}{x}$ untuk semua baris dan memilih baris dengan **rasio terendah**.

Table 9: Pemilihan Baris Pivot

x	y	u	w	P	b	rasio
3	5	1	0	0	78	26
4	1	0	1	0	36	9
-5	-4	0	0	1	0	0

STEP 2 Kita akan buat baris 2 kolom x menjadi bernilai 1, caranya dengan melakukan OBE seperti: $Row_2 = \frac{Row_2}{4}$.

Table 10: OBE Iterasi 1

x	y	u	w	P	b
3	5.00	1	0.00	0	78
1	0.25	0	0.25	0	9
-5	-4.00	0	0.00	1	0

STEP 3 Sekarang tujuan kita selanjutnya adalah membuat kolom x baris `1` dan `3` menjadi bernilai **nol**. Caranya adalah:

$$Row_1 = Row_1 - 3Row_2$$

$$Row_3 = Row_3 + 5Row_2$$

Table 11: OBE Iterasi 2

x	y	u	w	P	b
0	4.25	1	-0.75	0	51
1	0.25	0	0.25	0	9
0	-2.75	0	1.25	1	45

STEP 4 Kita akan lakukan hal yang sama pada *step 1*, yakni memilih kolom dengan negatif terbesar. Yakni kolom y. Lalu kita akan hitung rasio setiap baris dan akan memilih rasio paling rendah.

Table 12: Pemilihan Baris Pivot Kembali

x	y	u	w	P	b	rasio
0	4.25	1	-0.75	0	51	12.00000
1	0.25	0	0.25	0	9	36.00000
0	-2.75	0	1.25	1	45	-16.36364

STEP 5 Maka kita akan pilih baris 1 menjadi pivot. Kolom y pada baris 1 harus bernilai 1 sehingga kita harus membuat $Row_1 = \frac{4Row_1}{17}$.

Table 13: OBE Iterasi 3

x	y	u	w	P	b
0	1.00	0.2352941	-0.1764706	0	12
1	0.25	0.0000000	0.2500000	0	9
0	-2.75	0.0000000	1.2500000	1	45

STEP 6 Kita akan buat klom y di baris 2 dan 3 menjadi **nol** dengan cara:

$$Row_2 = Row_2 - \frac{Row_1}{4}$$

$$Row_3 = Row_3 + \frac{11Row_1}{4}$$

Table 14: OBE Iterasi 4

x	y	u	w	P	b
0	1	0.2352941	-0.1764706	0	12
1	0	-0.0588235	0.2941176	0	6

x	y	u	w	P	b
0	0	0.6470588	0.7647059	1	78

Dari tabel terakhir di atas, kita bisa menuliskan $x = 6, y = 12$ dan nilai $\max(P) = 78$. Bagaimana dengan nilau u dan w? Karena tidak ada nlai `1` ditemukan pada kolom variabel tersebut, kita bisa simpulkan bahwa $u = 0, w = 0$.

5.4 Metode *Branch and Bound* untuk *MILP*

Metode *simplex* adalah metode eksak yang digunakan untuk menyelesaikan *linear programming*. Solusi yang dihasilkan merupakan bilangan *real* atau kontinu. Pada *MILP*, variabel yang terlibat sangat beragam (*integer*, *binary*, dan kontinu). Membulatkan bilangan solusi *linear programming* untuk mendapatkan solusi *integer* atau *binary* dari suatu masalah *MILP* tidak menjamin keoptimalan tercapai.

Oleh karena itu, kita akan melakukan pendekatan tertentu dari *linear programming* agar hasilnya bisa digunakan di *MILP*.

5.4.1 *Relaxation of Discrete Optimization Models*

Salah satu pendekatan yang bisa dilakukan adalah melakukan *constraint relaxation* (Chachuat 2011).

Definisi Model R disebut dengan *constraint relaxation* dari model P jika:

- Setiap *feasible solution* dari P juga *feasible* di R.
- P dan R memiliki fungsi objektif yang sama.

Contoh Berikut adalah *original MILP*:

$$\min_{x,y} 7x_1 + x_2 + 3y_1 + 6y_2$$

$$\text{s.t. } x_1 + 10x_2 + 2y_1 + y_2 \geq 100$$

$$y_1 + y_2 \leq 1$$

$$x_1, x_2 \geq 0, y_1, y_2 \in \{0, 1\}$$

Relaxation I : *relax constraints RHS*

$$\min_{x,y} 7x_1 + x_2 + 3y_1 + 6y_2$$

$$\text{s.t. } x_1 + 10x_2 + 2y_1 + y_2 \geq 50$$

$$y_1 + y_2 \leq 1$$

$$x_1, x_2 \geq 0, y_1, y_2 \in \{0, 1\}$$

Relaxation II : *Drop constraint*

$$\min_{x,y} 7x_1 + x_2 + 3y_1 + 6y_2$$

$$\text{s.t. } x_1 + 10x_2 + 2y_1 + y_2 \geq 100$$

$$x_1, x_2 \geq 0, y_1, y_2 \in \{0, 1\}$$

Relaxation III : *remove integrality*

$$\min_{x,y} 7x_1 + x_2 + 3y_1 + 6y_2$$

$$\text{s.t. } x_1 + 10x_2 + 2y_1 + y_2 \geq 100$$

$$y_1 + y_2 \leq 1$$

$$x_1, x_2 \geq 0, 0 \leq y_1, y_2 \leq 1$$

5.4.2 *Linear Programming Relaxation*

Definisi *LP relaxation* dari *MILP* dibentuk dengan memperlakukan variabel diskrit sebagai variabel kontinu sambil mempertahankan semua *constraints* yang ada (Chachuat 2011).

$$y \in \{0, 1\} \Rightarrow 0 \leq y \leq 1$$

Oleh karena itu bisa terjadi hal sebagai berikut:

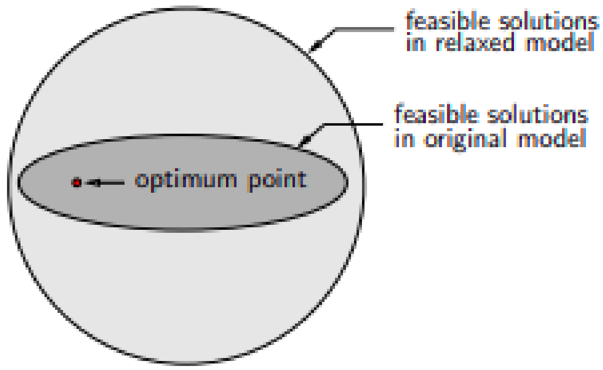

Figure 6: Solusi LP Relaxation

Apakah *LP relaxation* menjamin mendapatkan hasil yang *valid*?

Berikut adalah sifat dari *LP relaxation*:

1. Jika *LP relaxation infeasible*, maka model *MILP* asalnya juga.
2. Hasil optimal *LP relaxation* dari *MILP* yang bertujuan untuk maksimisasi berada pada *upper bound*.
3. Hasil optimal *LP relaxation* dari *MILP* yang bertujuan untuk minimisasi berada pada *lower bound*.
4. Jika suatu solusi optimal *LP relaxation* ternyata *feasible*, maka solusi tersebut optimal di model *MILP* asalnya.

5.4.3 Algoritma *Branch and Bound*

Algoritma *branch and bounds* mengkombinasikan beberapa strategi *relaxation* secara iteratif untuk memilih kemungkinan solusi paling optimal.

Ilustrasi Perhatikan contoh berikut:

$$\max z = 4x_1 - x_2$$

$$\text{s.t. } 7x_1 - 2x_2 \leq 14$$

$$x_2 \leq 3$$

$$2x_1 - 2x_2 \leq 3$$

$$x_1, x_2 \in \mathbb{Z}^+$$

Misalkan S adalah himpunan solusi *feasible* dari *LP relaxation* (dibuat suatu *LP relaxation* dengan x berupa variabel kontinu). Menggunakan metode simplex kita bisa dapatkan $x_1 = 2.857143, x_2 = 3, z = 8.428571$.

Kita misalkan $z* = -\infty$, karena x_1 bukan integer, maka kita akan uat *branch out* dari variabel ini.

$$S_1 = S \cap \{x : x_1 \leq 2\}$$

$$S_2 = S \cap \{x : x_1 \geq 3\}$$

Kita akan evaluasi kembali dengan *LP relaxation* yang baru.

- Pada S_1 kita dapatkan dengan metode *simplex* solusinya adalah $x_1 = 2, x_2 = 0.5, z = 0.75$. Oleh karena itu, kita akan *branch out* kembali dengan pemecahan sebagai berikut:

$$S_{11} = S \cap \{x : x_2 = 0\}$$

$$S_{12} = S \cap \{x : x_2 \geq 1\}$$

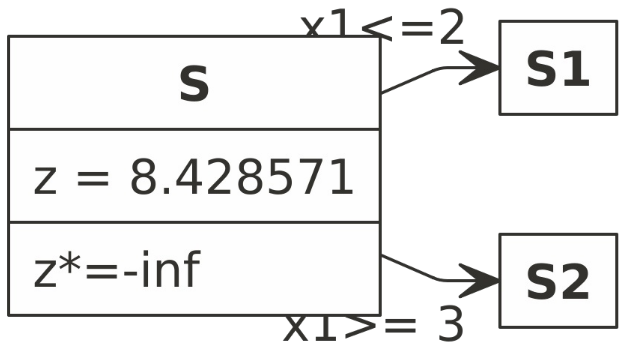

Figure 7: Branch Out Tahap I

- Pada S_2 kita dapatkan bahwa kondisi $x_1 \geq 3$ membuat model menjadi *infeasible*. Kita akan hentikan *branch out* dari S_2.

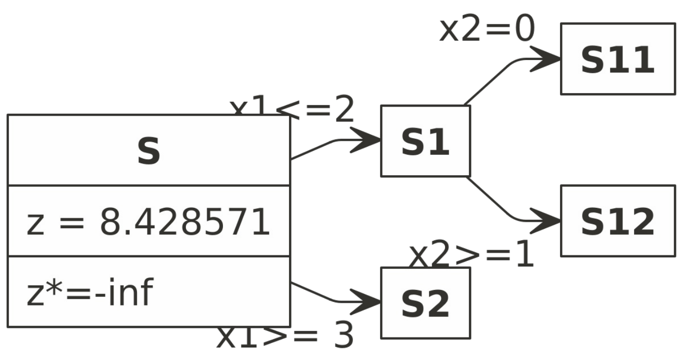

Figure 8: Branch Out Tahap II

Kita lakukan kembali *LP relaxation* pada S_{11} dan S_{12} sebagai berikut:

- Pada S_{12}, metode simplex menghasilkan solusi $x_1 = 2, x_2 = 1, z = 7$. Kita akan *update* nilai $z* = 7$.
- Pada S_{11}, metode simplex menghasilkan solusi $x_1 = 1.5, x_2 = 0, z = 6$. Karena $z < z*$, maka tidak ada lagi *branch out*.

Kesimpulan Solusi optimal didapatkan pada S_{12}.

5.5 R *Library* untuk Metode *Simplex*

Pada bagian sebelumnya, kita telah membahas bagaimana cara melakukan metode *simplex* dengan operasi baris elementer.

> Apakah proses operasi tersebut bisa diformalkan dalam bentuk algoritma atau **R** *codes*?

Salah satu dari sekian banyak *packages* yang memiliki *function* metode *simplex* yang siap pakai adalah `library(boot)` (Canty and Ripley 2021). Pada bagian ini, kita akan membahas salah satu *function* pada *library* tersebut.

Fungsi `simplex()` dari `library(boot)` digunakan untuk menyelesaikan masalah optimisasi linear ax dengan *constraints* $A_1x \leq b_1$, $A_2x \leq b_2$, $A_3x \leq b_3$, dan $x \geq 0$. Secara *default*, *function* ini akan mengukur *minimize*. Namun kita bisa mengubahnya menjadi *maximize*.

5.5.1 Penggunaan `simplex()`

Penggunaannya adalah sebagai berikut:

```
simplex(a, A1 = NULL, b1 = NULL, A2 = NULL, b2 = NULL, A3 = NULL,
        b3 = NULL, maxi = FALSE, n.iter = n + 2 * m, eps = 1e-10)
```

Dimana:

- `a` = *vector* yang merupakan koefisien dari *objective function*.
- `A1` = merupakan matriks berisi koefisien dari constraints bertipa $\leq$.
- `b1` = *vector* pasangan dari matriks `A1`. Harus berisi *non-negative*.
- `A2` = merupakan matriks berisi koefisien dari constraints bertipa $\geq$.
- `b2` = *vector* pasangan dari matriks `A2`. Harus berisi *non-negative*. Perhatikan bahwa *constraints* ≥ 0 secara *default* sudah masuk.
- `A3` = merupakan matriks berisi koefisien dari constraints bertipa $=$.

- `b3` = *vector* pasangan dari matriks `A3`. Harus berisi *non-negative*. Perhatikan bahwa *constraints* ≥ 0 secara *default* sudah masuk.
- `maxi` = *logical*, secara *default* akan mencari *minimize*.

Catatan Khusus Penulis *packages* ini memberikan suatu catatan khusus, yakni:

> *The method employed here is suitable only for relatively small systems.*

5.5.2 Contoh Penggunaan `simplex()`

Saya akan ambil contoh masalah *linear programming* sebagai berikut:

$$\text{maximize: } 7x_1 + 3x_2 + x_3$$

$$\text{subject to: } \begin{aligned} 6x_1 + 4x_2 + 5x_3 &\leq 60 \\ 8x_1 + x_2 + 2x_3 &\leq 80 \\ 9x_1 + x_2 + 7x_3 &\leq 70 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

```
rm(list=ls())
# panggil library
library(boot)

# set objective function
obj = c(7, 3, 1)
# membuat matriks A1
c1 = c(6, 4, 5)
c2 = c(8, 1, 2)
c3 = c(9, 1, 7)
A1 = rbind(c1,c2,c3)
```

```
# membuat rhs dari matriks A1
b1 = c(60, 80, 70)
# solving masalah optimisasi
simplex(a = obj, A1 = A1, b1 = b1,maxi = TRUE)
```

```
##
## Linear Programming Results
##
## Call : simplex(a = obj, A1 = A1, b1 = b1, maxi = TRUE)
##
## Maximization Problem with Objective Function Coefficients
## x1 x2 x3
##  7  3  1
##
##
## Optimal solution has the following values
##       x1       x2       x3
## 7.333333 4.000000 0.000000
## The optimal value of the objective  function is 63.3333333333333.
```

5.6 *Post-Optimality Analysis*

Post-optimality analysis adalah analisa yang dilakukan pasca kita telah menemukan *optimal solution* dari hasil perhitungan. Contohnya kita bisa melakukan **reoptimisasi**.

5.7 *Sensitivity Analysis*

Salah satu proses dalam membuat model optimisasi adalah *parameter estimation*. Ada kalanya perubahan data mengakibatnya berubahnya suatu parameter. *Sensitivity analysis* bertujuan untuk mengidentifikasi parameter yang sensitif (parameter yang harus dihitung dengan baik untuk menghindari kesalahan saat mencari solusi optimal).

6 R *PACKAGES* UNTUK OPTIMISASI

Untuk menyelesaikan masalah optimisasi menggunakan **R**, ada beberapa *packages* yang bisa digunakan. Saya akan bahas beberapa *packages* yang biasa digunakan untuk menyelesaikan masalah optimisasi di **R**, yakni:

1. `ROI` *packages* (Theußl, Schwendinger, and Hornik 2020).
2. `ompr` *packages* (Schumacher 2020).

6.1 `ROI` *Packages* di R

`ROI` merupakan singkatan dari **R Optimization Infrastructure** merupakan salah satu *packages* yang memberikan infrastruktur untuk menyelesaikan *linear programming*, *quadratic programming*, *conic*, dan *general non linear programming*.

`ROI` dikembangkan oleh **WU Vienna University of Economics and Business**[10], yakni:

- Kurt Hornik,
- David Meyer,
- Florian Schwendinger,
- Stefan Theussl,
- Diethelm Wuertz.

`ROI` bekerja dengan memanfaatkan berbagai *solver* (disebut dengan ***plugins***) yang dikembangkan oleh pihak-pihak lain. Dari masalah yang ada, kita **bisa melihat dan menentukan** *solver* apa yang bisa kita pakai untuk menyelesaikan permasalahan tersebut.

[10] https://epub.wu.ac.at/5858/

6.1.1 `ROI` *Modelling*

Framework untuk menuliskan model optimisasi menggunakan `ROI` adalah sebagai berikut:

Objective Function dimasukkan ke dalam *script* dengan format tergantung dari masalah yang dihadapi:

1. Jika berupa *linear programming*, *objective function* akan berupa `vector` numerik.
2. Jika berupa *quadratic programming*, *objective function* akan berupa `matriks`.

Constraints dalam `ROI` dimasukkan dalam bentuk pisahan berikut:

$$(parameter) + (direction) + (rhs)$$

Bounds atau batas *decision variables* termasuk tipenya (*integer* dan *kontinu*).

6.1.1.1 Contoh Penyelesaian *Linear Programming* I

$$\text{minimize: } 7x_1 + 8x_2$$

$$\text{subject to:} \quad \begin{aligned} 3x_1 + 4x_2 &= 9 \\ 2x_1 + x_2 &\geq 3 \\ -100 \leq x_1, x2 &\leq 100 \end{aligned}$$

Berikut adalah penyelesaian di **R**:

```
rm(list=ls())

# libraries dan solver yang digunakan
library(ROI)
```

```
library(ROI.plugin.glpk)
library(ROI.plugin.qpoases)
library(ROI.plugin.ecos)
library(ROI.plugin.scs)
library(ROI.plugin.alabama)
library(ROI.plugin.lpsolve)

# pendefinisian objective function
obj_func = L_objective(c(7, 8), names=c("x", "y"))
obj_func
```

```
## A linear objective of length 2.
```

```
# pendefinisian constraints
const = L_constraint(L = rbind(c(3, 4),
                               c(2, 1)),
                     dir = c("==", ">="),
                     rhs = c(9, 3)
                     )
const
```

```
## An object containing 2 linear constraints.
```

```
# pendefinisian bounds
bou = V_bound(li = 1:2, # x dan y
              ui = 1:2, # x dan y
              lb = c(-100, -100),
              ub = c(100, 100))
bou
```

```
## ROI Variable Bounds:
```

```
##
## 2 lower and 2 upper non-standard variable bounds.
```

```
# menggabungkan semua komponen
linear_problem = OP(objective = obj_func,
                    constraints = const,
                    bounds = bou)
linear_problem
```

```
## ROI Optimization Problem:
##
## Minimize a linear objective function of length 2 with
## - 2 continuous objective variables,
##
## subject to
## - 2 constraints of type linear.
## - 2 lower and 2 upper non-standard variable bounds.
```

Untuk melihat *solver* `ROI`, kita akan menggunakan perintah sebagai berikut:

```
ROI_applicable_solvers(linear_problem)
```

```
## [1] "glpk"    "alabama" "ecos"    "lpsolve" "qpoases" "scs"
```

Terlihat ada `6` *solvers* yang bisa dipilih. Proses mencari solusi dilakukan dengan perintah sebagai berikut:

```
ROI_solve(model ROI, solver = "nama solver")
```

Sebagai contoh, saya akan menggunakan *solver* `glpk`, maka:

```
solusi = ROI_solve(linear_problem,
                   solver = "glpk")
```

Berikut adalah hasilnya:

```
solution(solusi)
```

```
##   x   y
## 0.6 1.8
```

6.1.1.2 Contoh Penyelesaian *Linear Programming* II

$$\text{maximize: } 7x_1 + 3x_2 + x_3$$

$$\text{subject to: } \begin{aligned} 6x_1 + 4x_2 + 5x_3 &\leq 60 \\ 8x_1 + x_2 + 2x_3 &\leq 80 \\ 9x_1 + x_2 + 7x_3 &\leq 70 \\ x_1, x_2, x_3 &\geq 0 \end{aligned}$$

Berikut adalah penyelesaiannya di **R**:

```
linear_problem_2 =
  OP(# objective function
     objective = L_objective(c(7, 1, 3),
                             names = c("x", "y", "z")),
     # constraints
     constraints = L_constraint(L = rbind(c(6, 4, 5),
                                          c(8, 0, 2),
                                          c(9, 1, 7)
                                          ),
```

```
                                dir = c("<=", "<=", "<="),
                                rhs = c(60, 80, 70)),
     # tidak ada bounds yang perlu diset
     # pengaturan agar memaksimalkan obj function
     maximum = TRUE)

# proses pencarian solusi
# tanpa memanggil solver (default)
solusi_2 = ROI_solve(linear_problem_2)
# output
solution(solusi_2)
```

```
##        x        y        z
## 7.333333 4.000000 0.000000
```

Perlu diperhatikan pada baris-baris akhir, `linear_problem_2` diselesaikan tanpa kita harus memanggil *solver* yang ada. `ROI` akan memilihkan *default solver*.

6.1.1.3 Contoh Penyelesaian *Mixed Integer Linear Programming*

$$\text{maximize: } 7x_1 + 3x_2 + x_3$$

$$\text{subject to: } \begin{aligned} 6x_1 + 4x_2 + 5x_3 &\leq 60 \\ 8x_1 + x_2 + 2x_3 &\leq 80 \\ 9x_1 + x_2 + 7x_3 &\leq 70 \\ x_3 &\geq 0 \\ x_1, x_2 &\in \mathbb{Z}_{\geq 0} \end{aligned}$$

Masalah kali ini adalah perpaduan antara variabel kontinu dan *integer*.

Kita cukup memodifikasi perintah di **R** untuk mendefinisikan tipe variabel yang terlibat sebagai berikut:

```
mixed_ilp =
  OP(# objective function
     objective = L_objective(c(7, 1, 3),
                             names = c("x", "y", "z")),
     # constraints
     constraints = L_constraint(L = rbind(c(6, 4, 5),
                                          c(8, 0, 2),
                                          c(9, 1, 7)
                                          ),
                                dir = c("<=", "<=", "<="),
                                rhs = c(60, 80, 70)),
     # tidak ada bounds yang perlu diset

     # pendefinisian tipe variabel
     types = c("I","I","C"),
     # pengaturan agar memaksimalkan obj function
     maximum = TRUE)

# proses pencarian solusi
# tanpa memanggil solver (default)
solusi_3 = ROI_solve(mixed_ilp)
# output
solution(solusi_3)
```

```
##   x   y   z
## 7.0 4.0 0.4
```

6.1.2 *Conclusion*

Salah satu ciri khas dalam `ROI` adalah *input object* berupa **matriks** dan **vektor**.

6.2 `ompr` *Packages* di R

Ada satu *packages* lain di **R** yang bisa digunakan untuk menyelesaikan masalah optimisasi, yakni bernama `ompr`. *Packages* `ompr` dibuat oleh **Dirk Schumacher** pada 2018[11].

Salah satu keuntungan dari *library* ini adalah pengunaan operator *pipe* `%>%` pada perumusan algoritmanya. Sehingga bagi *user* yang biasa menggunakan prinsip `tidyverse` akan merasa sangat terbantu.

6.2.1 `ompr` *Modelling*

Framework untuk menuliskan model optimisasi menggunakan `ompr` adalah sebagai berikut:

```
# mulai membangun model
MIPModel() %>%

  # menambah variabel
  add_variable() %>%

  # set objective
  set_objective() %>%

  # menambah constraints
  add_constraint()
```

Decision Variable harus didefinisikan sejak awal. Ada berapa dan tipenya seperti apa. Kita bisa menggunakan *indexed variables* untuk menghemat notasi. Berikut adalah contohnya:

[11] https://www.r-orms.org/

```
MIPModel() %>%

  # menambah variabel integer
  add_variable(x, type = "integer") %>%

  # menambah variabel kontinu
  add_variable(y, type = "continuous") %>%

  # menambah variabel binary integer
  add_variable(z, type = "binary") %>%

  # menambah variabel dengan lower bound
  add_variable(x, lb = 10) %>%

  # menambah variabel dengan upper dan lower bounds
  add_variable(y, lb = 5, ub = 10) %>%

  # menambah 10 variabel berindeks
  add_variable(p[i], i = 1:10)
```

Objective Function* dan *Constraints dalam `ompr` bisa dituliskan sebagai fungsi matematika biasa. Bahkan kita bisa menuliskan *summation* ke dalam algoritmanya. Berikut adalah contohnya:

Misal ada 3 variabel x_1, x_2, x_3, dengan *objective function* $\sum_i x_i$ dengan *constraint* $\sum_i x_i \leq 7$.

```
MIPModel() %>%
  add_variable(x[i], i = 1:3) %>%
  set_objective(sum_expr(x[i], i = 1:3)) %>%
  add_constraint(sum_expr(x[i], i = 1:3) <= 7)
```

6.2.1.1 Contoh Penyelesaian *Mixed Integer Linear Programming*

$$\text{maximize: } 7x_1 + 3x_2 + x_3$$

$$\text{subject to: } \begin{aligned} 6x_1 + 4x_2 + 5x_3 &\leq 60 \\ 8x_1 + x_2 + 2x_3 &\leq 80 \\ 9x_1 + x_2 + 7x_3 &\leq 70 \\ x_3 &\geq 0 \\ x_1, x_2 &\in \mathbb{Z}_{\geq 0} \end{aligned}$$

Mari kita tuliskan dalam `ompr` *framework* berikut:

```
rm(list=ls())

# memanggil libraries
library(dplyr)
library(ompr)
library(ompr.roi)
library(ROI.plugin.glpk)

# membuat model
milp_new =
  MIPModel() %>%

  # membuat 2 variabel integer
  add_variable(x1,type = "integer",lb = 0) %>%
  add_variable(x2,type = "integer",lb = 0) %>%

  # membuat 1 variabel kontinu
  add_variable(x3,type = "continuous",lb = 0) %>%
```

```
  # set obj function
  set_objective(7*x1 + 3*x2 + x3,
                "max") %>%

  # menuliskan semua constraints
  add_constraint(6*x1 + 4*x2  + 5*x3 <= 60) %>%
  add_constraint(8*x1 + x2 + 2*x3 <= 80) %>%
  add_constraint(9*x1 + x2 + 7*x3 <= 70)

milp_new
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 1
##   Integer: 2
##   Binary: 0
## Model sense: maximize
## Constraints: 3
```

Mari kita *solve* modelnya:

```
result = solve_model(milp_new, with_ROI(solver = "glpk", verbose = TRUE))
```

```
## <SOLVER MSG>  ----
## GLPK Simplex Optimizer, v4.65
## 3 rows, 3 columns, 9 non-zeros
## *     0: obj =  -0.000000000e+00 inf =   0.000e+00 (3)
## *     2: obj =   6.333333333e+01 inf =   0.000e+00 (0)
## OPTIMAL LP SOLUTION FOUND
## GLPK Integer Optimizer, v4.65
```

```
## 3 rows, 3 columns, 9 non-zeros
## 2 integer variables, none of which are binary
## Integer optimization begins...
## Long-step dual simplex will be used
## +     2: mip =     not found yet <=              +inf        (1; 0)
## +     4: >>>>>   6.140000000e+01 <=   6.166666667e+01   0.4% (2; 0)
## +     4: mip =   6.140000000e+01 <=     tree is empty   0.0% (0; 3)
## INTEGER OPTIMAL SOLUTION FOUND
## <!SOLVER MSG> ----
```

```
result
```

```
## Status: optimal
## Objective value: 61.4
```

Berikut adalah hasilnya:

```
result %>% get_solution(x1)
```

```
## x1
##  7
```

```
result %>% get_solution(x2)
```

```
## x2
##  4
```

```
result %>% get_solution(x3)
```

```
##  x3
## 0.4
```

6.2.2 *Conclusion*

Salah satu ciri khas `ompr` adalah penulisannya yang mirip dengan notasi matematika sehingga saat kita memiliki suatu model dengan banyak variabel, kita tidak perlu menginputnya ke dalam bentuk matriks.

6.2.3 Penyelesaian Contoh Soal 3.2.1

Dengan menggunakan `library(ompr)`

```
rm(list=ls())

# memanggil libraries
library(dplyr)
library(ompr)
library(ompr.roi)
library(ROI.plugin.glpk)

# membuat model
integer_prog =
  MIPModel() %>%
  # membuat variabel
  add_variable(x[i],
               type = "integer",
               lb = 0,
               i = 1:7) %>%
  # set fungsi objective
  set_objective(sum_expr(x[i],i = 1:7),"min") %>%
  # memasukkan constraints
  # senin
  add_constraint(sum_expr(x[i], i = c(1,4,5,6,7)) >= 24) %>%
```

```
  add_constraint(sum_expr(x[i], i = c(1,4,5,6,7)) <= 29) %>%
  # selasa
  add_constraint(sum_expr(x[i], i = c(1,2,5,6,7)) >= 22) %>%
  add_constraint(sum_expr(x[i], i = c(1,2,5,6,7)) <= 27) %>%
  # rabu
  add_constraint(sum_expr(x[i], i = c(1,2,3,6,7)) >= 23) %>%
  add_constraint(sum_expr(x[i], i = c(1,2,3,6,7)) <= 28) %>%
  # kamis
  add_constraint(sum_expr(x[i], i = c(1,2,3,4,7)) >= 11) %>%
  add_constraint(sum_expr(x[i], i = c(1,2,3,4,7)) <= 16) %>%
  # jumat
  add_constraint(sum_expr(x[i], i = 1:5) >= 16) %>%
  add_constraint(sum_expr(x[i], i = 1:5) <= 21) %>%
  # sabtu
  add_constraint(sum_expr(x[i], i = 2:6) >= 20) %>%
  add_constraint(sum_expr(x[i], i = 2:6) <= 25) %>%
  # minggu
  add_constraint(sum_expr(x[i], i = 3:7) >= 12) %>%
  add_constraint(sum_expr(x[i], i = 3:7) <= 17)

integer_prog
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 0
##   Integer: 7
##   Binary: 0
## Model sense: minimize
## Constraints: 14
```

```r
hasil = integer_prog %>% solve_model(with_ROI(solver = "glpk",verbose = T))
```

```
## <SOLVER MSG>  ----
## GLPK Simplex Optimizer, v4.65
## 14 rows, 7 columns, 70 non-zeros
##       0: obj =   0.000000000e+00 inf =   1.280e+02 (7)
##       9: obj =   2.766666667e+01 inf =   0.000e+00 (0)
## *    10: obj =   2.766666667e+01 inf =   0.000e+00 (0)
## OPTIMAL LP SOLUTION FOUND
## GLPK Integer Optimizer, v4.65
## 14 rows, 7 columns, 70 non-zeros
## 7 integer variables, none of which are binary
## Integer optimization begins...
## Long-step dual simplex will be used
## +    10: mip =     not found yet >=              -inf        (1; 0)
## +    12: >>>>>   2.800000000e+01 >=   2.800000000e+01   0.0% (2; 0)
## +    12: mip =   2.800000000e+01 >=     tree is empty   0.0% (0; 3)
## INTEGER OPTIMAL SOLUTION FOUND
## <!SOLVER MSG> ----
```

```r
# solusi yang dihasilkan
hasil$solution
```

```
## x[1] x[2] x[3] x[4] x[5] x[6] x[7]
##    8    3    1    0    4   12    0
```

Kita telah mendapatkan konfigurasi jadwal nakes yang optimal perharinya.

6.2.4 Penyelesaian Contoh Soal 3.3.1

Dengan menggunakan `library(ompr)`

```
rm(list=ls())

library(dplyr)
library(ompr)
library(ompr.roi)
library(ROI.plugin.glpk)

bin_prog =
  MIPModel() %>%
  # menambah variabel
  add_variable(x[i,j],
           i = 1:20,
           j = 1:5,
           type = "binary",
           lb = 0) %>%
  # membuat objective function
  set_objective(sum_expr(x[i,j],
             i = 1:20,
             j = 1:5),
        "max") %>%
  # menambah constraints
  # max kapasitas kelas
  add_constraint(sum_expr(x[i,j],i = 1:20) >= 4,
         j = 1:5) %>%
  add_constraint(sum_expr(x[i,j],i = 1:20) <= 8,
         j = 1:5) %>%
  # frek kunjungan siswa
  add_constraint(sum_expr(x[i,j],j = 1:5) >= 2,
         i = 1:20) %>%
```

```
  add_constraint(sum_expr(x[i,j],j = 1:5) <= 3,
         i = 1:20) %>%
  # jeda sehari
  add_constraint(x[i,j] + x[i,j+1] <= 1,
         i = 1:20,
         j = 1:4)

bin_prog
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 0
##   Integer: 0
##   Binary: 100
## Model sense: maximize
## Constraints: 130
```

Berikut adalah hasilnya:

```
## <SOLVER MSG>  ----
## GLPK Simplex Optimizer, v4.65
## 130 rows, 100 columns, 560 non-zeros
##       0: obj =  -0.000000000e+00 inf =   6.000e+01 (25)
##      51: obj =   4.000000000e+01 inf =   0.000e+00 (0)
## *    53: obj =   4.000000000e+01 inf =   0.000e+00 (0)
## OPTIMAL LP SOLUTION FOUND
## GLPK Integer Optimizer, v4.65
## 130 rows, 100 columns, 560 non-zeros
## 100 integer variables, all of which are binary
## Integer optimization begins...
```

```
## Long-step dual simplex will be used
## +    53: mip =     not found yet <=              +inf        (1; 0)
## +    53: >>>>>   4.000000000e+01 <=   4.000000000e+01   0.0% (1; 0)
## +    53: mip =   4.000000000e+01 <=     tree is empty   0.0% (0; 1)
## INTEGER OPTIMAL SOLUTION FOUND
## <!SOLVER MSG> ----
```

Table 15: Jadwal Kunjungan Siswa

hari	presensi
1	1,2,3,4,9,10,11,12
2	5,6,7,8,13,14,15,16
3	1,2,3,4,17,18,19,20
4	5,6,7,8,13,14,15,16
5	9,10,11,12,17,18,19,20

Table 16: Rekap Presensi Siswa

siswa	jumlah kehadiran
1	2
2	2
3	2
4	2
5	2
6	2
7	2
8	2
9	2
10	2

siswa	jumlah kehadiran
11	2
12	2
13	2
14	2
15	2
16	2
17	2
18	2
19	2
20	2

6.2.5 Penyelesaian Contoh Soal 3.4.2

Dengan menggunakan `library(ompr)`

```
rm(list=ls())

library(dplyr)
library(ompr)
library(ompr.roi)
library(ROI.plugin.glpk)

# data yang dibutuhkan
profit = c(5,7,3)
sales = c(7,5,9)
M = 99999

# membuat model
mil_prog =
```

```
MIPModel() %>%
# menambah variabel
# xi
add_variable(x[i],
         i = 1:3,
         type = "continuous",
         lb = 0) %>%
# yi
add_variable(y[i],
         i = 1:3,
         type = "binary",
         lb = 0) %>%
# z
add_variable(z,type = "binary",lb = 0) %>%
# membuat objective function
set_objective(sum_expr(x[i] * profit[i],
           i = 1:3),
      "max") %>%
# menambah constraints
# max tonase
add_constraint(x[i] <= sales[i],
       i = 1:3) %>%
# memilih 2 produk
add_constraint(x[i] - y[i] * M <= 0,
       i = 1:3) %>%
add_constraint(sum_expr(y[i],
       i = 1:3) <= 2) %>%
# memilih 1 plant
add_constraint(3*x[1] + 4*x[2] + 2*x[3] - M * z <= 30) %>%
```

```
  add_constraint(4*x[1] + 6*x[2] + 2*x[3] + M * z <= 40 + M)


mil_prog
```

```
## Mixed integer linear optimization problem
## Variables:
##   Continuous: 3
##   Integer: 0
##   Binary: 4
## Model sense: maximize
## Constraints: 9
```

```
hasil =
  mil_prog %>%
  solve_model(with_ROI(solver = "glpk",
                       verbose = T))
```

```
## <SOLVER MSG>  ----
## GLPK Simplex Optimizer, v4.65
## 9 rows, 7 columns, 20 non-zeros
## *     0: obj =  -0.000000000e+00 inf =   0.000e+00 (3)
## *     7: obj =   9.700000000e+01 inf =   0.000e+00 (0)
## OPTIMAL LP SOLUTION FOUND
## GLPK Integer Optimizer, v4.65
## 9 rows, 7 columns, 20 non-zeros
## 4 integer variables, all of which are binary
## Integer optimization begins...
## Long-step dual simplex will be used
## +     7: mip =     not found yet <=              +inf        (1; 0)
## +    12: >>>>>   5.450000000e+01 <=   5.450000000e+01   0.0% (4; 0)
```

```
## +    12: mip =   5.450000000e+01 <=     tree is empty   0.0% (0; 7)
## INTEGER OPTIMAL SOLUTION FOUND
## <!SOLVER MSG> ----
```

```
xi =
  hasil %>%
  get_solution(x[i])

yi =
  hasil %>%
  get_solution(y[i])

zi =
  hasil %>%
  get_solution(z)
```

Berikut adalah hasilnya:

```
##   variable i value
## 1        x 1   5.5
## 2        x 2   0.0
## 3        x 3   9.0

##   variable i value
## 1        y 1     1
## 2        y 2     0
## 3        y 3     1

## z
## 1
```

Dari ketiga produk baru, perusahaan bisa memilih produk **1 dan 3** sebanyak **5.5 dan 9 ton** di *plant* **2**. Maka *profit* yang bisa diraih adalah sebesar **54.5**.

7 KOMPLEKSITAS ALGORITMA

Sebuah masalah bisa diselesaikan dengan berbagai macam algoritma. Sebuah algoritma tidak hanya diharuskan **benar** tapi juga **efisien**. Efisiensi suatu algoritma biasanya diukur dari:

1. Waktu eksekusi algoritma (*runtime*).
2. Kebutuhan *memory* komputasi (*memory allocation*).

Algoritma yang baik adalah algoritma yang meminimumkan kebutuhan waktu dan *memory* (Munir 2015).

Kebutuhan waktu dan *memory* dari suatu algoritma bergantung pada ukuran *input* (n) yang menyatakan jumlah data yang diproses.

Definisi Besaran yang dipakai untuk mengukur waktu dan *memory* ini disebut **kompleksitas algoritma**.

7.1 Perhitungan Kompleksitas Waktu

Menghitung waktu *real* dari eksekusi algoritma tidak bisa dilakukan karena bisa jadi ada perbedaan dalam hal:

1. Spesifikasi perangkat keras yang digunakan.
2. Spesifikasi perangkat lunak yang digunakan.

Definisi Perhitungan kompleksitas waktu $T(n)$ diukur dari tahapan komputasi yang dibutuhkan untuk menjalankan algoritma sebagai fungsi dari *input* n.

Sebagai contoh, suatu algoritma yang digunakan untuk menghitung rata-rata dari suatu data $\{1, 2, .., n\}$ sebagai berikut:

```
sum = 0
```

```
for i in 1 to n:
  sum = sum + i
avg = sum / n
```

Memiliki kompleksitas waktu $T(n) = n$. Dihitung dari operasi mendasar di dalamnya yakni `sum = sum + i` yang diulang sebanyak n kali.

Kompleksitas waktu dibedakan menjadi tiga macam:

1. **Kebutuhan waktu maksimum** terjadi saat $T(n) = \max n$.
2. **Kebutuhan waktu minimum** terjadi saat $T(n) = \min n$.
3. **Kebutuhan waktu rata-rata** terjadi saat $T(n) = \text{avg } n$

7.2 Kompleksitas Waktu Asimptotik

Kompleksitas waktu asimptotik dinotasikan sebagai O (O-besar).

Definisi $T(n) = O(f(n))$ dibaca: $T(n)$ berorde paling besar $f(n)$ bila terdapat konstanta C dan n_0 sedemikian sehingga $T(n) \leq f(n)$ untuk $n \geq n_0$.

Kompleksitas bisa berupa konstan, logaritmik, linear, kuadratik, kubik, atau eksponensial[12].

7.2.1 Macam-macam Kompleksitas O

Konstan $O(k)$ dengan k suatu nilai tertentu yang tetap. Artinya algoritma ini membutuhkan k langkah dan tidak tergantung dari berapa banyak *input* n.

Linear $O(n)$ artinya algoritma ini akan berjalan sebanyak n langkah mengikuti *input*-nya.

Logaritmik $O(log(n))$ artinya algoritma ini membutuhkan $log(n)$ langkah.

[12] https://introprogramming.info/english-intro-csharp-book/read-online/chapter-19-data-structures-and-algorithm-complexity/

Kuadratik $O(n^2)$ artinya algoritma ini membutuhkan n^2 langkah.

Kubik $O(n^3)$ artinya algoritma ini membutuhkan n^3 langkah.

Eksponensial $O(k^n)$ untuk suatu nilai k tertentu. Artinya algoritma ini membutuhkan k^n langkah.

References

Canty, Angelo, and B. D. Ripley. 2021. *Boot: Bootstrap r (s-Plus) Functions*.

Cavazzuti, Marco. 2013. *Deterministic Optimization*. Berlin, Heidelberg: Springer-Verlag. https://doi.org/10.1007/978-3-642-31187-1_4.

Chachuat, Benoit. 2011. "MILP: Branch-and-Bound Search." http://macc.mcmaster.ca/maccfiles/chachuatnotes/07-MILP-I_handout.pdf.

Geovanni, Luigi De, and Marco Di Summa. 2018. "Methods and Models for Combinatorial Optimization: Heuristis for Combinatorial Optimization." https://www.math.unipd.it/~luigi/courses/metmodoc1819/m02.meta.en.partial01.pdf.

Hillier, Frederick S., and Gerald J. Lieberman. 2001. *Introduction to Operations Research*. 7th ed. New York, US: McGraw Hill. www.mhhe.com.

Kania, Adhe, and Kuntjoro Adji Sidarto. 2016. "Solving Mixed Integer Nonlinear Programming Problems Using Spiral Dynamics Optimization Algorithm." http://dx.doi.org/10.1063/1.4942987.

Lin, Ming-Hua, Jung-Fa Tsai, and Chian-Son Yu. 2012. "A Review of Deterministic Optimization Methods in Engineering and Management." https://doi.org/10.1155/2012/756023.

Munir, Rinaldi. 2015. *Catatan Kuliah Matematika Diskrit: Kompleksitas Algoritma*. Institut Teknologi Bandung.

Parmono, Vincentius Rachmadi. 2007. *Pengenalan Riset Operasi*. 1st ed. Jakarta, Indonesia: Universitas Terbuka. https://www.pustaka.ut.ac.id/lib/adbi4530-riset-operasi/.

Rothlauf, Franz. 2011. *Design of Modern Heuristics: Principles and Application*. 1st ed. Berlin, Germany: Springer. https://link.springer.com/book/10.1007/978-3-540-72962-4#about.

Schumacher, Dirk. 2020. "ompr: Model and Solve Mixed Integer Linear Programs."

Taha, Hamdy A. 2007. *Operations Research an Introduction*. 8th ed. New Jersey, US: Prentice Hall. https://link.springer.com/book/10.1007/978-3-540-72962-4#about.

Theußl, Stefan, Florian Schwendinger, and Kurt Hornik. 2020. "ROI: An Extensible R Optimization Infrastructure." *Journal of Statistical Software* 94 (15): 1–64. https://doi.org/10.18637/jss.v094.i15.