STUDI KASUS PERMASALAHAN *SET COVER* DENGAN *GREEDY* DAN *BRANCH & BOUND*

Disusun Untuk Memenuhi Tugas Mata Kuliah CII2K3-Strategi Algoritma

Disusun Oleh:

GAYUH PUTRI A.	1301204050
ALIF FAIDHIL AHMAD	1301204141
RAYHAN FADHIL R.	1301204501

PROGRAM STUDI INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY
BANDUNG
2022

DAFTAR ISI

ABSTRAK

Set cover problem merupakan salah satu persoalan optimasi dengan tujuan untuk menentukan koleksi *subset* sehingga semua elemen pada himpunan semesta termasuk di dalamnya dan koleksi *subset* tersebut memiliki kardinalitas atau bobot minimum jika persoalannya adalah *weighted set cover problem*. *Set Covering Problem* merupakan masalah optimasi populer yang telah diterapkan pada berbagai bidang seperti industri, telekomunikasi, biologi, dan farmasi. *Set cover problem* memiliki banyak algoritma penyelesaian. Algoritma yang terpopuler dan sering dibahas antara lain algoritma genetika, algoritma *Greedy*, dan algoritma *Branch & Bound*. Pada pembahasan kali ini menggunakan dua algoritma, yaitu algoritma *Greedy* dan algoritma *Branch & Bound*. Prinsip algoritma *Greedy* pada tiap langkah adalah mengambil pilihan yang terbaik pada saat itu tanpa memperhatikan konsekuensi ke depan dan berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global. Algoritma ini tidak selalu memberikan hasil terbaik, tetapi pasti memberikan yang optimal. Sedangkan algoritma *Branch & Bound* selalu memberikan solusi terbaik karena membandingkan semua solusi yang mungkin dan mengambil salah satu yang memiliki total *cost* terendah.

Kata kunci: *greedy, branch & bound, set cover problem*

BAB I
PENDAHULUAN

1.1 Pengertian Algoritma

Menurut (Kami, 2020, 1.19), algoritma adalah suatu upaya dengan urutan operasi yang disusun secara logis dan sistematis untuk menyelesaikan suatu masalah untuk menghasilkan suatu *output* tertentu. Algoritma berasal dari kata *algoris* dan ritmis yang pertama kali diperkenalkan oleh Abu Ja'far Muhammad Ibn Musa Al Khwarizmi pada 825 M di dalam buku "Al-Jabr Wa-al Muqabla". Dalam bidang pemrograman, algoritma didefinisikan sebagai metode yang terdiri dari serangkaian langkah yang terstruktur dan sistematis untuk menyelesaikan masalah dengan bantuan komputer (Jando & Nani, 2018, 5). Sedangkan menurut (Munir & Lidya, 2016, 5), algoritma adalah urutan langkah-langkah untuk menyelesaikan suatu persoalan. Algoritma juga dapat diartikan sebagai sekumpulan instruksi yang terstruktur dan terbatas yang diimplementasikan kedalam bentuk program komputer untuk menyelesaikan suatu masalah komputasi tertentu. Dalam matematika dan ilmu komputer, algoritma adalah prosedur langkah-demi-langkah untuk penghitungan (Sismoro, 2005, 29).

1.2 Pengertian Strategi

Secara bahasa strategi berasal dari kata *strategic* yang berarti menurut siasat atau rencana dan *strategy* yang berarti ilmu siasat. Menurut istilah strategi adalah rencana yang cermat mengenai kegiatan untuk mencapai sasaran khusus. Selain itu, strategi dapat disebut sebagai suatu pendekatan yang semuanya berhubungan dengan implementasi, perencanaan, dan pelaksanaan ide dalam kegiatan dengan jangka waktu tertentu. Menurut Norton dan Kaplan, strategi adalah kumpulan hipotesis dalam hubungan sebab-akibat, yaitu hubungan yang dapat diekspresikan oleh hubungan antara *if* dan *then*. Berdasarkan pengertian tersebut dapat disimpulkan bahwa strategi adalah suatu proses yang direncanakan untuk mencapai sasaran dalam jangka waktu yang panjang. Saat strategi telah diterapkan maka akan diketahui apakah gagal atau berhasil pada suatu masalah.

1.3 Pengertian Strategi Algoritma dan Klasifikasinya

Strategi algoritma adalah suatu teknik atau metode yang digunakan dalam pembentukan algoritma atau aturan langkah langkah untuk memecahkan suatu persoalan agar tercapai solusi dari persoalan tersebut. Algoritma dikatakan baik jika *running time*, *time complexity* dan total *cost*-nya kecil. Untuk mengetahui algoritma dikatakan baik atau tidak, dapat dilakukan analisis strategi mana yang sebaiknya digunakan dalam suatu persoalan yaitu dengan melihat seberapa besar *time complexity* dari algoritma terhadap input yang diberikan (Wijanarto, repository.dinus.ac.id).

Strategi Algoritma dapat diklasifikasikan menjadi empat. Pertama, Strategi solusi langsung (*direct solution strategies*), contohnya adalah Algoritma *Brute Force* dan Algoritma *Greedy*. Kedua, Strategi berbasis pencarian pada ruang status (*state-space base strategies*), seperti pada Algoritma *Backtracking* dan Algoritma *Branch and Bound*. Ketiga, Strategi solusi atas-bawah (*top-down solution strategies*), misalnya pada Algoritma *Divide and Conquer*, Algoritma *Decrease and Conquer*, dan *Dynamic Programming*. Dan yang terakhir adalah Strategi solusi bawah-atas (*bottom-up solution strategies*) dengan contohnya adalah *Dynamic Programming*. (Diktat kuliah Strategi Algoritmik, Rinaldi Munir)

1.4 Pengertian *Set Cover*

SCV (*Set Covering Problem*) adalah masalah optimasi populer yang telah diterapkan pada berbagai aplikasi industri, termasuk penjadwalan, manufaktur, perencanaan layanan, dan masalah lokasi. Rumusan matematika dari SCP (*Set Covering Problem*) adalah sebagai berikut. U = {x1, …, xm} sebagai anggota himpunan semesta, S = {s1, …, sn} yang merupakan subset dari U, tujuannya adalah untuk mencari subset Sn C S dengan kardinalitas minimum, sehingga semua elemen pada U terdapat pada subset 'S'. Setiap set ini meng-cover setidaknya satu elemen dari U dan memiliki *cost* > 0. Tujuannya adalah untuk menemukan sub lokasi himpunan yang mencakup semua elemen dengan *cost* seminimal mungkin. Solusi untuk SCP (*Set Covering Problem*) dianggap tidak layak jika satu atau lebih elemen pada himpunan semesta ditemukan. Suatu himpunan dianggap redundan jika semua elemen yang tercakup dalam himpunan tersebut juga tercakup oleh himpunan lain dalam penyelesaian.

BAB II
DASAR TEORI

2.1 Studi Kasus

Set cover problem merupakan salah satu persoalan optimasi dengan tujuan untuk menentukan koleksi *subset* sehingga semua elemen pada himpunan semesta termasuk di dalamnya dan koleksi subset tersebut memiliki kardinalitas atau bobot minimum jika persoalannya adalah *weighted set cover problem*. *Set Covering Problem* merupakan masalah optimasi populer yang telah diterapkan pada berbagai bidang seperti industri, telekomunikasi, biologi, dan farmasi.

Pada pembahasan kali ini, kami mengambil kasus *set cover* problem di bidang industri farmasi. Diceritakan bahwa terdapat seorang ilmuwan laboratorium penelitian farmasi. Untuk menunjang kebutuhan penelitiannya di laboratorium tersebut, maka ia perlu membeli satu di antara banyak set peralatan. Setelah menimbang beberapa hal dan melakukan analisis, ia menyimpulkan bahwa market menawarkan sejumlah paket peralatan penunjang penelitian, karena peralatan ini merupakan produk luar, masing-masing berharga sekian euro. Setiap paket produk menawarkan beberapa fungsi dan memberikan keunggulannya masing-masing. Namun sayangnya, tidak semua paket setara. Pada suatu paket terdapat keunggulan yang sangat menguntungkan, tetapi beberapa terdapat suatu hal yang merugikan atau kurang memuaskan. Oleh karena itu, tujuan kami adalah untuk membantu untuk memutuskan paket peralatan mana yang direkomendasikan untuk dibeli. Selain itu, untuk meminimalisir pengeluaran maka kami juga memperhitungkan biaya yang dikeluarkan.

2.2 Pemilihan Strategi Algoritma

Set cover problem memiliki banyak algoritma penyelesaian. Algoritma yang terpopuler dan sering dibahas antara lain algoritma genetika, algoritma *greedy*, dan algoritma *branch & bound*. Pada pembahasan kali ini menggunakan dua algoritma, yaitu algoritma *greedy* dan algoritma *branch & bound*.

2.2.1 *Greedy*

Algoritma *Greedy* banyak digunakan untuk mengatasi masalah prioritas pada *test-case*, yang berfokus pada pemilihan *test-case* terbaik saat ini selama penentuan tersebut. Algoritma *Greedy* diklasifikasikan menjadi dua kelompok. Yang pertama, bertujuan untuk memilih tes yang mencakup lebih banyak statement. Sedangkan yang kedua bertujuan untuk memilih tes yang terjauh dari tes yang dipilih.

Perihal kelompok pertama, algoritma *Greedy* yang paling populer adalah algoritma total dan algoritma tambahan. Secara khusus, algoritma total memprioritaskan kasus uji berdasarkan urutan pernyataan yang dicakup oleh setiap kasus uji, sedangkan algoritma tambahan memprioritaskan kasus uji berdasarkan urutan pernyataan yang dicakup oleh setiap kasus uji yang tidak dipilih tetapi ditemukan oleh yang dipilih yang ada. Kedua algoritma ini dapat memiliki kinerja terbaik dalam kasus yang berbeda, Zhang et al. [174, 175]

Perihal kelompok kedua, algoritma *Greedy* yang khas adalah prioritas kasus uji acak adaptif [177], yang diusulkan berdasarkan pengujian acak adaptif [178, 179]. Algoritma pemilihan bertujuan untuk memilih kasus uji yang terjauh dari kasus uji yang sudah dipilih berdasarkan fungsi definisi jarak f1 dan fungsi pemilihan terjauh f2. Secara khusus, karya ini mengusulkan untuk menggunakan jarak Jaccard untuk mendefinisikan f1 dan mendefinisikan tiga jenis fungsi seleksi f2.

Algoritma *Greedy* berfokus pada pencarian solusi optimal lokal untuk prioritas, dan dengan demikian hasil prioritas mereka mungkin bukan solusi optimal.

2.2.2 *Branch & Bound*

Algoritma *Branch & Bound* merupakan salah satu contoh dari kelas Strategi berbasis pencarian pada ruang status (*state-space base strategies*). Selain itu, algoritma ini merupakan salah satu algoritma yang paling banyak digunakan dalam optimasi, dalam istilah sederhananya disebut sebagai tulang punggung pemrograman integer campuran.

Algoritma ini secara berturut-turut menghasilkan bagian dari pohon solusi dan menghitung kriteria. Kapan pun suboptimal urutan parsial atau *node* ditemukan memenuhi, *subtree* di bawah *node* secara implisit ditolak, dan pencacahan dimulai pada urutan parsial yang belum dieksplorasi.

Faktanya algoritma *Branch & Bound* secara bertahap memberikan solusi yang lebih baik dengan batas yang lebih ketat. Ini berarti bahwa suatu algoritma, ketika diakhiri lebih awal, terkadang juga memberikan solusi yang berarti. Hal tersebut sering digunakan dalam optimasi untuk bootstrap beberapa metode lain dengan titik awal yang berarti.

BAB III
IMPLEMENTASI

Secara konsep matematika, permasalahan ini dapat diformulasikan sebagai berikut:

$$\text{minimize } \sum_{set \epsilon S} W_{\text{set}} * x_{\text{set}} \text{ subject to } \sum_{set:e \epsilon S} x_{\text{set}} \geq 1 \quad \text{Where } \text{x}_{\text{set}} \epsilon \{0,1\}$$

artinya suatu himpunan dapat ada atau tidak ada dalam cakupan *subset*. 'S' adalah cakupan *set*, 'W' adalah *list cost* yang ditetapkan, 'U' adalah himpunan semesta dan 'e' adalah elemen pada himpunan semesta. Jadi permasalahan ini membutuhkan minimalisasi *total cost* dengan mempertimbangkan bahwa setiap elemen 'e' harus tercakup setidaknya sekali.

3.1 Penerapan Algoritma *Greedy*

Prinsip algoritma *Greedy* pada tiap langkah adalah mengambil pilihan yang terbaik pada saat itu tanpa memperhatikan konsekuensi ke depan dan berharap bahwa dengan memilih optimum lokal pada setiap langkah akan berakhir dengan optimum global.
Elemen-elemen dalam algoritma *Greedy*:

1. Himpunan kandidat, yang berisi elemen-elemen pembentuk solusi.
2. Himpunan solusi, berisi kandidat-kandidat yang terpilih sebagai solusi persoalan.
3. Fungsi seleksi, dinyatakan dengan predikat seleksi memilih kandidat yang paling memungkinkan mencapai solusi optimal pada setiap langkah.
4. Fungsi kelayakan, dinyatakan dengan predikat layak, memeriksa apakah suatu kandidat yang telah dipilih dapat memberikan solusi yang layak dengan tidak melanggar constraints yang ada.
5. Fungsi objektif, yang memaksimumkan atau meminimumkan nilai solusi.

Dengan kata lain, algoritma *Greedy* melibatkan pencarian sebuah himpunan bagian, S, dari himpunan kandidat, C; yang dalam hal ini, S harus memenuhi beberapa kriteria yang ditentukan, yaitu menyatakan suatu solusi dan S dioptimisasikan oleh fungsi objektif. Sesuai dengan prinsip *Greedy* yang memilih optimum lokal pada setiap langkah, maka di setiap langkah pada algoritma *greedy-set-cover*, fungsi seleksi akan memilih subset yang menutupi jumlah elemen yang belum ditutup paling banyak.

Ide utama dari algoritma ini adalah untuk menutupi himpunan semesta dengan mengambil setiap kali set yang tampaknya paling meyakinkan dalam list set. Dengan kata lain: setiap siklus *while* program akan mencari di antara semua himpunan dan akan mengambil satu dengan rasio tertinggi antara elemen yang belum tercakup dan *cost* relatif dari himpunan. Algoritma ini tidak selalu memberikan hasil terbaik, tetapi pasti memberikan yang optimal.

Pada setiap langkah algoritma *greedy-set-cover*, akan dipilih *subset* S dari himpunan subset F. *Subset* yang dipilih tersebut adalah *subset* yang memaksimalkan |S n U| atau dengan kata lain *subset* yang mampu menutupi elemen yang belum ditutupi dengan jumlah paling banyak. Setelah S dipilih, elemen anggota U yang dapat ditutupi dengan *subset* S dihapus dan *subset* ditambahkan menjadi anggota himpunan *subset* C. Langkah ini terus diulangi sampai semua elemen pada X ditutupi atau U adalah himpunan kosong.

3.2 Penerapan Algoritma *Branch & Bound*

Metode ini menggunakan pohon pencarian (*Search Tree*), setiap simpul di pohon merupakan representasi dari sejumlah kemungkinan solusi suatu masalah. Algoritma dimulai dengan pengisian sebuah nilai ke akar dari pohon pencarian tersebut. Pencabangan dilakukan dengan memasang sebuah pending *node* ke pending *node* lain yang lebih rendah levelnya. Bobot juga dihitung pada setiap proses dan ditulis di simpul pohon. Jika sebuah simpul diketahui merupakan solusi yang tidak mungkin bagi persoalan yang dihadapi, simpul tersebut diisi dengan nilai tak terbatas (*infinity*). Algoritma berhenti ketika sudah tidak mungkin lagi untuk membentuk simpul baru di pohon atau hasil terakhir yang ditemukan merupakan hasil yang lebih rendah (minimum) dari isi simpul yang telah ada pada level yang lebih rendah

Pada penerapannya, *branch* (cabang) pada algoritma ini dikembangkan selama simpulnya tidak menjadi lebih besar dari *cost* terbaik yang ditemukan sejauh ini dan elemen-elemen dalam himpunan terpilih membentuk seluruh himpunan semesta. Ada dua fungsi tambahan disini, yaitu *bypass branch* (pemotongan cabang), yang hanya melewati solusi yang tidak layak atau tidak optimal; dan *next vertex* (simpul berikutnya), fungsi ini penting untuk terus membuat himpunan bagian baru. Algoritma ini selalu memberikan solusi terbaik karena membandingkan semua solusi yang mungkin dan mengambil salah satu yang memiliki total cost terendah.

BAB IV
ANALISIS

4.1 Algoritma *Greedy*

4.1.1 *Pseudocode* Algoritma *Greedy*

```
SetCover(Universe,Subsets,Costs)
    cost ← 0
    elements ← all elements in subsets taken just one time
    if elements != universe:
        return
    covered ← []
    cover ← []
    while covered != elements:
        subset ← set with the highest 'elements not in covered'/'cost' ratio
         cover ← cover + subset
         cost ← cost + cost of subset
         covered ← covered + element in subset and not in covered
     return cover,cost
```

4.1.2 Kompleksitas Waktu Algoritma *Greedy*

$T(n) = O (log_e n)$

4.2 Algoritma *Branch & Bound*

4.2.1 *Pseudocode* Algoritma *Branch & Bound*

```
bypassbranch(subset,i)
    for j ← i to 1
        if subset_j = 0:
            subset_j ← 1
            return subset,j+1
    return subset,0
```

```
nextvertex(subset,i,m)
    if i < m:
        subset_i ← 0
        return subset,i+1
    else:
        for j ← m to 1
            if subset_j = 0:
                subset_j ← 1
                return subset,j+1
     return subset,0

SetCover(universe,sets,costs)
    subset ← [1]* |sets|
    subset_1 ← 0
    bestcost ← sum(costs)
    i ← 2
    while i > 0:
        if i < |sets|:
            cost ← 0
            tset ← []
             for k ← 1 to i:
                cost ← cost + subset_k * costs_k
                if subset_k = 1:
                    tset ← tset + sets_k
             if cost > bestcost :
                 subset,i ← bypassbranch(subset,i)
             else:
                 subset,i ← nextvertex(subset,i,|sets|)
         else:
             cost ← 0
             fset ← []
             for k ← 1 to i:
                 cost ← cost + subset_k * costs_k
                if subset_k = 1:
                    fset ← fset + sets_k
                if cost < bestcost and elements in fset = universe:
                    bestcost ← cost
                    bestsubset ← subset
            subset,i ← nextvertex(subset,i,|sets|)
     return bestcost,bestsubset
```

4.2.2 Kompleksitas Waktu Algoritma *Branch & Bound*

$T(n) = O(f(n))$

4.3 Perbandingan Waktu Algoritma *Greedy* dan *Branch & Bound*

4.3.1 Perbandingan Waktu dari Algoritma *Greedy*

Nama	Nilai Maksimum	Jumlah *Sub Set*	*Running Time*
m1	5	10	0,005
m2	15	45	0,089
m3	5	21	0,005
m4	40	23	0,027
m5	30	23	0,04

4.3.2 Perbandingan Waktu dari Algoritma *Branch & Bound*

Nama	Nilai Maksimum	Jumlah *Sub Set*	*Running Time*
m1	5	10	0,003
m2	15	45	0,06
m3	5	21	0,005
m4	40	23	0,024
m5	30	23	0,055

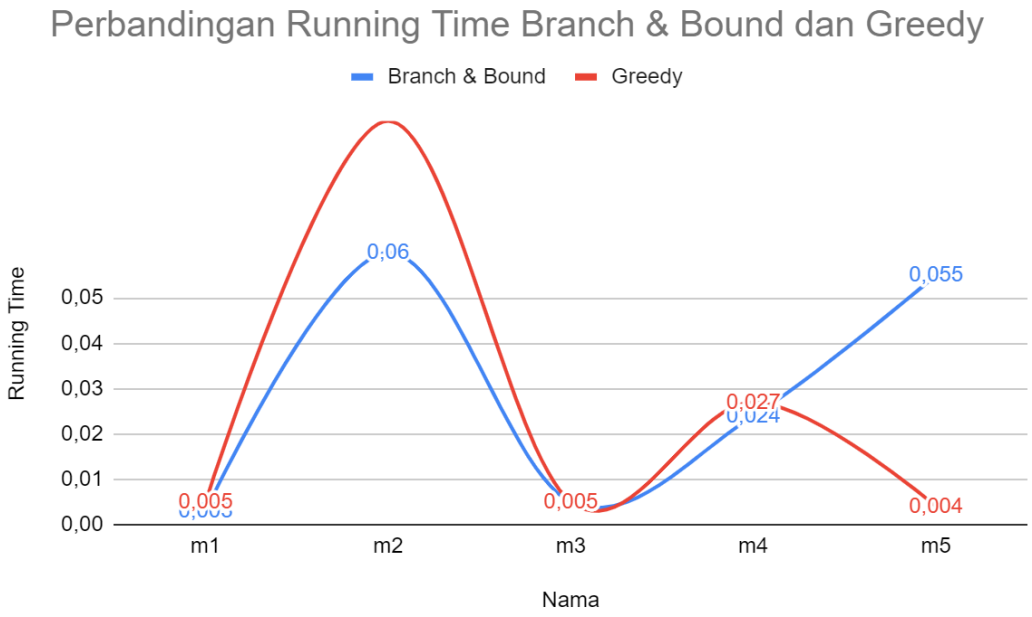
4.3.3 Perbandingan Waktu dari Kedua Algoritma

Pada dasarnya, algoritma *Greedy* dapat menemukan solusi cukup mudah untuk suatu masalah dan dapat menganalisis *running time* jauh lebih mudah. Sedangkan algoritma *Branch & Bound* merupakan metode yang lambat, karena adanya kompleksitas waktu eksponensial selama *worst case*. Tetapi, algoritma ini terkadang menjadi lebih efisien dan membantu untuk menentukan optimasi global dalam masalah *non-convex*.

Pada hasil percobaan yang telah kami lakukan, didapatkan *running time* yang cenderung tidak konsisten. Beberapa contoh yang diuji memiliki nilai *running time* untuk *Branch & Bound* lebih cepat dibandingkan *Greedy* dan beberapa yang lainnya justru kebalikannya. Tetapi ada satu contoh pengujian yang memiliki nilai running time yang sama pada kedua algoritma. Hal ini dapat terjadi karena *subset* dan atau himpunan semesta pada pengujian bersifat random. Selain itu, algoritma *Branch & Bound* tidak menutup kemungkinan untuk lebih efisien pada kasus tertentu. Sehingga untuk mendapatkan hasil pengujian yang lebih optimal membutuhkan lebih banyak pengujian dengan berbagai macam *subset* dan atau himpunan semesta pada kasus *set cover* ini.

Nama	*Running Time*	
	Branch & Bound	***Greedy***
m1	0,003	0,005

m2	0,06	0,089
m3	0,005	0,005
m4	0,024	0,027
m5	0,055	0,04



BAB V
KESIMPULAN

Berdasarkan permasalahan studi kasus tersebut, kami melakukan percobaan, pengujian, dan analisis menggunakan dua metode algoritma, yaitu *Greedy* dan *Branch & Bound*. Kompleksitas waktu setiap algoritma, yaitu

- *Greedy* = O ($log_e n$)
- *Branch & Bound* = O ($f(n)$)

Berdasarkan analisis, penyelesaian dapat menggunakan kedua algoritma tersebut. Dengan menggunakan *Greedy*, dapat dengan lebih mudah untuk mendapatkan solusi, tetapi tidak selalu menghasilkan solusi optimum global. Sedangkan, *Branch & Bound* terkadang menjadi lebih efisien walaupun metode ini dapat cenderung lambat. Menurut pandangan kami, untuk permasalahan studi kasus ini lebih baik diselesaikan menggunakan metode algoritma *Greedy*. Karena kasus *set cover problem* dapat memiliki kemungkinan terdiri dari banyak *subset* dan himpunan semesta yang ada. Demi keefisienan waktu yang digunakan agar lebih cepat dalam prosesnya.

DAFTAR PUSTAKA

Yiling Lou, Junjie Chen, Lingming Zhang, Dan Hao, 2019, “Chapter One - A Survey on Regression Test-Case Prioritization”

Nehme Bilal, Philippe Galinier, and Francois Guibault, 2013, “A New Formulation of the Set Covering Problem for Metaheuristic Approaches”

Marin Vlastelica Pogančić, 2019, “The Branch and Bound Algorithm”

Eka Mukti Arifah, 2009, “Penyelesaian Set Cover Problem dengan Algoritma Greedy”

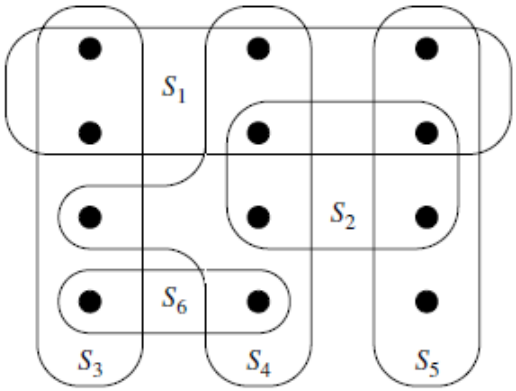
Darsih Idayani, Yesi Puspitasari, Lisma Dian Kartika Sari, 2020, “Penggunaan Model Set Covering Problem dalam Penentuan Lokasi dan Jumlah Pos Pemadam Kebakaran”

John M. Echols, Hassan Shadily, Kamus Inggris Indonesia, Jakarta: PT Gramedia Pustaka Utama, 2007, h. 701.

Tim Penyusun, Kamus Besar Bahasa Indonesia, ed. 3 cet. 3, Jakarta: Balai Pustaka 2005, h. 423

LAMPIRAN

Contoh *Set Cover*



Source Code

Source Code dapat diakses melalui *link* berikut:
https://github.com/alifahmad14/Studi-Kasus-Permasalahan-Set-Cover-dengan-Greedy-dan-Branch-Bound

sourcecode.py

```python
#Import library untuk membantu menghasilkan waktu/durasi
import time

#Fungsi Set Cover; mengecek setiap elemen pada subset
def set_cover(universe, subsets,costs):
    cost = 0
    elements = set(e for s in subsets for e in s)
    if elements != universe:
        return None
    covered = set()
    cover = []
    while covered != elements:
        subset = max(subsets, key=lambda s: len(s - covered)/costs[subsets.index(s)])
        cover.append(subset)
        cost += costs[subsets.index(subset)]
        covered |= subset

    return cover, cost
```

sourcecode.py

```
#fungsi Main pada Algoritma Greedy; untuk menjalankan algoritma tersebut
def Gmain(a,b,c):
    m= a
    universe = set(range(1, m+1))
    sub = b

    subsets = [set(x) for x in sub]
    costs =  c
    cover = set_cover(universe, subsets,costs)
    print('covering sets= ',cover[0],'\n', 'cost= ',cover[1],'$')
```

sourcecode.py

```
#Branch and Bound
#Fungsi untuk melakukan bypass
def bypassbranch(subset, i):
    for j in range(i-1, -1, -1):
        if subset[j] == 0:
            subset[j] = 1
            return subset, j+1

    return subset, 0
```

sourcecode.py

```python
#Fungsi membuat himpunan new subsets
def nextvertex(subset, i, m):
    if i < m:
        subset[i] = 0
        return subset, i+1
    else:
        for j in range(m-1, -1, -1):
            if subset[j] == 0:
                subset[j] = 1
                return subset, j+1

    return subset, 0
```

sourcecode.py

```python
#Fungsi Branch and Bound sesuai rumus
def BB(universe,sets,costs):
    subset = [1 for x in range(len(sets))]
    subset[0] = 0
    bestCost = sum(costs) #Worst cost
    i = 1

    while i > 0:

        if i < len(sets):
            cost, tSet = 0, set()# t = temporary
            for k in range(i):
                cost += subset[k]*costs[k]
                if subset[k] == 1: tSet.update(set(sets[k]))

            if cost > bestCost:
                subset, i = bypassbranch(subset, i)
                continue
            for k in range(i, len(sets)): tSet.update(set(sets[k]))
            if tSet != universe:
                subset, i = bypassbranch(subset, i)
            else:
                subset, i = nextvertex(subset, i, len(sets))

        else:
            cost, fSet = 0, set()
            for k in range(i):
                cost += subset[k]*costs[k]
                if subset[k] == 1: fSet.update(set(sets[k]))

            if cost < bestCost and fSet == universe:
                bestCost = cost
                bestSubset = subset[:]
            subset, i = nextvertex(subset, i , len(sets))

    return bestCost, bestSubset
```

sourcecode.py

```
#fungsi Main pada Algoritma Branch and Bound; untuk menjalankan algoritma
tersebut
def BBmain(a,b,c):
    m = a
    S = b
    C = c
    F = set([x for x in range(1,m+1)])
    X=(BB(F,S,C))
    cost= X[0]
    sets= X[1]
    cover= []
    for x in range(len(sets)):
        if sets[x]==1:
            cover.append(S[x])

    print('covering sets: ',cover,'\n','total cost: ',cost,'$')
```

sourcecode.py

```
#Inisialisasi inputan
#10 set
m1= 5
S1 = [[1,3],[2],[1,2,5],[3,5],[4],[5],[1,3],[2,4,5],[1,2],[2,3]]
P1 = [11,4,9,12,5,4,13,12,8,9]

#45 set
m2 = 15
S2 = [[2, 7, 8, 10, 12, 13], [1, 3, 5, 8, 10, 11, 12, 15], [1, 2, 3, 4, 5, 6, 7, 12, 13],
[2, 6, 7, 11, 12, 13], [9, 10, 12, 13], [1, 3, 7, 9, 11, 12, 13], [1, 3, 5, 6, 8, 9, 10, 11, 12, 13],
[1, 3, 4, 5, 6, 7, 12, 14, 15], [1, 2, 3, 6, 11, 12], [1, 2, 4, 5, 7, 8], [5, 9, 10, 11, 15],
[3, 5, 6, 7, 8, 9, 12, 13, 14], [1, 3, 4, 5, 6, 7, 9, 11, 13, 14, 15], [1, 3, 5, 6, 8, 12, 14],
[2, 4, 7, 9, 10, 12, 14], [1, 3, 5, 6, 11, 15], [2, 3, 4, 5, 6, 8, 10, 11, 12, 13, 14, 15],
[1, 2, 4, 6, 7, 11, 13, 14, 15], [1, 2, 8, 12, 13, 14], [1, 2, 6, 7, 8, 13],
[1, 2, 3, 5, 7, 8, 10, 12, 14, 15], [4, 5, 7, 12, 15], [1, 2, 3, 5, 11, 14], [1, 6, 8, 11, 13],
[1, 6, 7, 8, 9, 10, 13], [1, 2, 3, 4, 5, 9, 11, 15], [2, 3, 4, 7, 9, 11, 12],
[1, 3, 4, 5, 8, 10, 11, 12, 13], [2, 8, 9, 10], [6, 11, 13], [2, 5, 6, 8, 9, 11, 12, 13, 15],
[2, 4, 6, 7, 8, 9, 10, 11, 13, 15], [1, 2, 3, 4, 5, 7, 8, 10, 11], [1, 2, 6, 9, 11, 13, 14, 15],
[1, 4, 9, 10, 11, 13, 15], [1, 2, 3, 4, 6, 8, 12, 14, 15], [4, 5, 7, 8, 10, 13, 14],
[2, 4, 8, 9, 11, 14], [2, 3, 4, 5, 6, 7, 10, 11, 14], [1, 2, 4, 5, 6, 7, 9, 11, 13, 14, 15],
[1, 2, 6, 7, 9, 10, 12, 15], [1, 3, 6, 9, 10, 15], [2, 3, 5, 7, 8, 9, 11],
[2, 3, 4, 5, 8, 10, 11, 12, 15], [1, 3, 4, 5, 6, 7, 9, 10, 12, 15]]
P2 = [16, 7, 16, 39, 29, 35, 19, 27, 27, 33, 38, 8, 41, 16, 12, 7, 41, 6, 34, 48, 23, 16, 31, 18, 35, 31, 41, 21, 50, 21,
12, 37, 35, 44, 48, 18, 14, 26, 22, 13, 29, 34, 28, 45, 50]

#21 set
m3 = 5
S3 = [[1], [1, 2, 3, 4, 5], [2, 3], [2, 3, 4, 5], [1, 3, 4], [5], [1, 2, 4], [1, 3, 4, 5], [3, 5],
[4, 5], [3], [2, 5], [4], [1, 5], [2], [1, 2, 4], [1, 3], [1, 3, 5], [2, 4, 5], [2], [1, 2, 5]]
P3 = [44, 44, 39, 24, 5, 30, 26, 42, 28, 12, 6, 45, 37, 33, 5, 42, 26, 6, 38, 11, 28]
```

sourcecode.py

```
#Inisialisasi inputan
#23 set
m4 = 40
S4 = [[1, 3, 4, 6, 7, 9, 10, 15, 16, 18, 26, 31, 32, 35, 36, 38, 39,
40],
[1, 2, 3, 4, 5, 7, 9, 11, 13, 15, 17, 18, 19, 20, 23, 24, 25, 27, 31,
32, 37, 39, 40],
[4, 7, 8, 10, 11, 14, 16, 17, 18, 20, 23, 24, 27, 28, 29, 34, 36, 37,
39, 40],
[2, 3, 4, 7, 9, 11, 17, 20, 22, 25, 26, 27, 28, 32, 34, 35, 36, 37, 39,
40],
[1, 2, 4, 6, 7, 10, 12, 13, 22, 23, 24, 26, 28, 30, 32, 33, 35, 36,
39],
[1, 3, 4, 5, 6, 8, 9, 10, 12, 13, 16, 24, 25, 30, 34, 35, 36, 37, 38,
39],
[2, 3, 5, 10, 11, 12, 14, 18, 20, 22, 24, 25, 27, 28, 30, 31, 33, 34,
40],
[1, 3, 11, 12, 18, 19, 21, 22, 24, 25, 26, 30, 33, 35],
[1, 2, 7, 9, 10, 11, 14, 16, 18, 20, 22, 25, 28, 33, 35, 38],
[3, 4, 9, 10, 14, 15, 17, 18, 19, 23, 24, 26, 28, 29, 30, 32, 34, 35,
38, 39, 40],
[1, 2, 3, 4, 5, 6, 7, 9, 10, 13, 14, 15, 16, 19, 20, 22, 23, 29, 30,
31, 36, 38, 39],
[2, 4, 5, 7, 13, 14, 15, 17, 20, 23, 24, 25, 27, 28, 29, 30, 34, 35],
[1, 2, 4, 8, 9, 11, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24, 26, 27, 31,
32, 33, 34, 35, 36, 37, 39, 40],
[1, 4, 5, 6, 8, 10, 14, 17, 20, 21, 23, 24, 25, 29, 30, 40],
[3, 5, 6, 10, 12, 14, 16, 17, 18, 19, 20, 22, 23, 24, 26, 28, 29, 30,
31, 32, 33, 34, 37, 39],
[2, 3, 5, 6, 7, 9, 14, 15, 16, 17, 20, 21, 23, 27, 28, 29, 31, 32, 34,
35, 39, 40],
[2, 5, 7, 10, 11, 13, 14, 18, 20, 22, 23, 29, 32, 33, 34, 35, 38, 39],
[1, 3, 6, 7, 8, 9, 10, 12, 13, 24, 29, 30, 33, 34, 35, 36, 37, 39, 40],
[1, 2, 5, 6, 8, 9, 10, 11, 12, 13, 14, 16, 20, 21, 26, 29, 30, 32, 33,
35, 36, 38, 39, 40],
[3, 4, 7, 8, 11, 14, 16, 17, 19, 20, 21, 22, 23, 24, 25, 26, 29, 30,
31, 33, 34, 36, 38, 39],
[2, 3, 4, 6, 7, 9, 11, 13, 14, 15, 16, 19, 21, 24, 25, 26, 27, 28, 29,
30, 31, 33, 36, 39],
[1, 2, 3, 6, 8, 10, 11, 13, 15, 16, 17, 19, 20, 21, 22, 25, 26, 34, 35,
36, 39, 40],
[1, 2, 7, 12, 15, 17, 21, 24, 25, 27, 28, 30, 35, 37, 39, 40]]
P4 = [59, 68, 56, 50, 75, 95, 71, 66, 30, 28, 42, 50, 68, 34, 29, 52,
70, 85, 27, 40, 76, 82, 38]
```

sourcecode.py

```
#Inisialisasi inputan
#23 set
m5 = 30
S5 = [[2, 3, 4, 8, 9, 10, 11, 12, 15, 16, 18, 19, 22, 23, 24, 26, 27,
28, 29],
[1, 2, 4, 5, 7, 8, 10, 11, 13, 16, 17, 19, 20, 21, 22, 23, 25, 27, 30],
[1, 3, 9, 10, 16, 17, 18, 23, 24, 25, 26, 29, 30],
[2, 4, 5, 6, 7, 10, 11, 12, 13, 14, 17, 20, 21, 22, 26, 27, 29, 30],
[1, 6, 7, 11, 14, 17, 18, 23, 25, 26, 28, 29],
[3, 5, 6, 7, 9, 10, 12, 13, 15, 16, 17, 18, 19, 21, 22, 24, 25, 26,
29],
[2, 5, 6, 7, 8, 9, 10, 13, 17, 19, 20, 21, 23, 24, 25, 27, 28, 29, 30],
[1, 5, 6, 8, 10, 19, 21, 24], [1, 3, 7, 8, 9, 10, 15, 19, 25, 26, 27,
30],
[4, 5, 10, 11, 12, 13, 14, 16, 18, 20, 21, 22, 27, 28, 29], [1, 6, 7,
9, 17, 23, 26, 29, 30],
[3, 4, 7, 8, 12, 13, 14, 15, 19, 21, 22, 24, 25, 27, 28, 29, 30],
[1, 5, 6, 7, 8, 10, 15, 19, 21, 22, 27, 28, 29, 30],
[1, 2, 4, 5, 6, 7, 8, 9, 11, 13, 14, 17, 19, 20, 23, 25, 26, 28, 30],
[2, 3, 4, 9, 12, 15, 17, 20, 23, 26, 27, 28, 29],
[4, 7, 8, 9, 11, 12, 13, 14, 15, 16, 19, 20, 25, 26, 27, 28, 29, 30],
[1, 3, 5, 7, 8, 9, 18, 19, 20, 21, 22, 23, 26, 28, 29],
[1, 3, 6, 7, 8, 10, 12, 13, 14, 16, 21, 23, 24, 25, 26, 27],
[3, 5, 8, 9, 12, 13, 15, 18, 20, 21, 23, 24, 29],
[1, 3, 4, 5, 8, 9, 10, 14, 15, 16, 18, 19, 20, 21, 22, 24, 26, 27],
[2, 5, 8, 9, 12, 13, 14, 15, 17, 19, 20, 21, 24, 27, 28, 30],
[2, 4, 8, 9, 12, 15, 16, 23, 24, 27, 28], [4, 5, 9, 10, 11, 12, 14, 15,
19, 22, 23, 27, 30]]
P5 = [60, 79, 49, 65, 88, 83, 38, 44, 54, 100, 65, 53, 43, 73, 63, 35,
65, 92, 74, 79, 67, 34, 95]
```

sourcecode.py

```
#MAIN PROGRAM
#Output
print('SET COVER PROBLEM USING BRANCH AND BOUND AND GREEDY ALGORITHM')

print('\n', 'BRANCH AND BOUND ALGORITHM')
print('\n', 'm1')
a = time.time()
BBmain(m1,S1,P1)
a1 = time.time()-a
print('time:',a1)

print('\n', 'm2')
b = time.time()
BBmain(m2,S2,P2)
b1 = time.time()-b
print('time:',b1)

print('\n', 'm3')
c=time.time()
BBmain(m3,S3,P3)
c1 = time.time()-c
print('time:',c1)

print('\n', 'm4')
d = time.time()
BBmain(m4,S4,P4)
d1 = time.time()-d
print ('time:',d1)

print('\n', 'm5')
e = time.time()
BBmain(m5,S5,P5)
e1 = time.time()-e
print('time:',e1)
```

sourcecode.py

```
#MAIN PROGRAM
#Output

print('\n', 'GREEDY ALGORITHM')
print('\n', 'm1')
aa = time.time()
BBmain(m1,S1,P1)
a2 = time.time()-aa
print('time:',a2)

print('\n', 'm2')
bb = time.time()
BBmain(m2,S2,P2)
b2 = time.time()-bb
print('time:',b2)

print('\n', 'm3')
cc=time.time()
BBmain(m3,S3,P3)
c2 = time.time()-cc
print('time:',c2)

print('\n', 'm4')
dd = time.time()
BBmain(m4,S4,P4)
d2 = time.time()-dd
print('time:',d2)

print('\n', 'm5')
ee = time.time()
BBmain(m5,S5,P5)
e2 = time.time()-ee
print('time:',e2)
```

sourcecode.py

```
#MAIN PROGRAM
#Output

print('\n', 'Perbedaan Running time', '\n')
print('m1','\n','Branch and Bound: ', a1, '\n', 'Greedy: ', a2, '\n')
print('m2','\n','Branch and Bound: ', b1, '\n', 'Greedy: ', b2, '\n')
print('m3','\n','Branch and Bound: ', c1, '\n', 'Greedy: ', c2, '\n')
print('m4','\n','Branch and Bound: ', d1, '\n', 'Greedy: ', d2, '\n')
print('m5','\n','Branch and Bound: ', e1, '\n', 'Greedy: ', e2, '\n')
```