Penyelesaian *Set Cover Problem* dengan Algoritma *Greedy*

Eka Mukti Arifah – NIM : 13507100

Jurusan Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha No. 10 Bandung
e-mail: if17100@students.if.itb.ac.id

ABSTRAK

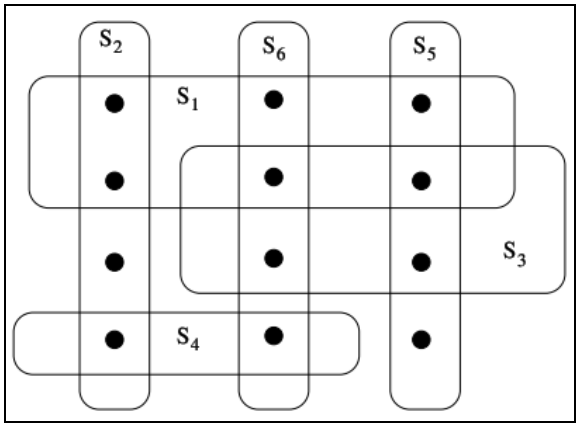
Makalah ini membahas tentang penyelesaian *set cover problem* dengan algoritma *greedy*. *Set cover problem* adalah salah satu persoalan optimasi dimana tujuannya adalah menentukan koleksi subset sehingga semua elemen pada himpunan semesta termasuk di dalamnya dan koleksi subset tersebut memiliki kardinalitas atau bobot minimum jika persoalannya adalah *weighted set cover problem*. *Set cover problem* memiliki banyak algoritma penyelesaian. Algoritma yang paling sering dibahas antara lain algoritma genetik, algoritma *greedy*, dan algoritma *branch and bound*. Pada makalah ini hanya dibahas mengenai penyelesaian *set cover problem* dengan menggunakan algoritma *greedy*. Prinsip algoritma *greedy* adalah memilih pilihan optimum pada setiap langkah agar langkah selanjutnya mengarah ke solusi optimum global. Algoritma *greedy* memang tidak selalu memberikan solusi optimum global, namun algoritma ini adakah algoritma aproksimasi *set cover problem* dengan waktu polinomial yang paling baik.

Kata kunci: *Set Cover Problem*, algoritma *greedy*, optimasi

1. PENDAHULUAN

Set cover problem (SCP) adalah salah satu contoh persoalan optimasi. *Set cover problem* didefinisikan sebagai berikut. Terdapat semesta $U = \{x_1, x_2, x_3, ..., x_n\}$ dan koleksi $S = \{S_1, S_2, S_3, ..., S_m\}$ yang merupakan subset dari *U*, tujuannya adalah untuk mencari subset $S' \subseteq S$ dengan kardinalitas minimum sehingga semua elemen pada *U* terdapat pada subset *S'* atau $\cup_{S_j \in S'} S_j = U$. Pada *weighted set covering problem*, setiap set S_i diberikan bobot bernilai positif w_i, dan dicari *S'* yang memberikan jumlah bobot minimum dari $S_i \in S'$ atau $\sum_{i=1}^{m} w_i s_i$ minimum dimana s_i = 1 jika s_i terpilih dalam solusi dan s_i = 0 jika s_i tidak terpilih.

Set cover optimization problem merupakan persoalan NP-hard klasik. Algoritma penyelesaian SCP antara lain algoritma genetik, algoritma *greedy*, dan algoritma *branch and bound*. Algoritma *greedy* memberikan solusi aproksimasi logaritmik. Algoritma *greedy* memang tidak selalu memberikan solusi optimum global untuk SCP, namun algoritma ini pada dasarnya adalah algoritma aproksimasi set cover problem dengan waktu polinomial yang paling baik.



Gambar 1. Contoh *set cover problem*

Set cover problem dapat diaplikasikan di berbagai bidang seperti telekomunikasi, biologi, dan farmasi. Secara praktikal, SCP digunakan antara lain untuk memilih lokasi penempatan *base station* jaringan selular, penempatan lokasi fasilitas darurat, dan penjadwalan kru penerbangan.

2. ALGORITMA *GREEDY*

Algoritma *greedy* merupakan algoritma yang paling populer untuk memecahkan masalah optimasi. Persoalan optimasi ada 2 macam, yaitu maksimasi dan minimasi.

Algoritma *greedy* membentuk solusi langkah per langkah. Pada setiap langkah, terdapat banyak pilihan yang perlu dieksplorasi. Dari pilihan tersebut, dipilih yang merupakan nilai optimum pada langkah tersebut (optimum lokal) dengan harapan bahwa langkah selanjutnya akan mengarah ke optimum global.

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. Himpuan 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* dioptimisasi oleh fungsi obyektif.

Berikut ini adalah skema umum algoritma *greedy*,

```
function greedy (input C : h_kandidat) → h_kandidat
{
   Mengembalikan solusi dari persoalan optimasi dengan algoritma greedy.
   Masukan : himpunan kandidat C.
   Keluaran : himpunan solusi yang bertipe himpunan_kandidat.
}

Deklarasi
   x : kandidat
   S : h_kandidat

   function SELEKSI(C : h_kandidat) → kandidat
   {me-return sebuah kandidat yang dipilih dari C berdasarkan kriteria tertentu}

   function SOLUSI(S : h_kandidat) → boolean
   {true jika S adalah solusi dari persoalan;
   false jika S belum menjadi solusi}

   function LAYAK(S : h_kandidat) → boolean
   {true jika S merupakan solusi yang tidak melanggar kendala;
   false jika S melanggar kendala}

Algoritma
   S ← {}  {inisialisasi S dengan kosong}
   while (not SOLUSI(S)) and (C ≠ {}) do
      x ← SELEKSI(C){pilih 1 kandidat dari C}
      C ← C - {x}   {elemen himpunan kandidat berkurang 1}
      if LAYAK (S ∪{x}) then
         S ← S ∪ {x}
      endif
   endwhile
   {SOLUSI(S) or C ={} }

   if SOLUSI(S) then
      return S
   else
      write ('tidak ada solusi')
   endif
```

3. ALGORITMA *GREEDY-SET-COVER*

2.1 Skema Algoritma *Greedy-Set-Cover*

Algoritma ini merupakan algoritma *greedy* untuk menyelesaikan *set cover problem*. Seperti yang sudah disebutkan sebelumnya, pada *set cover problem* terdapat semesta $U = \{x_1, x_2, x_3, ..., x_n\}$ dan koleksi $S = \{S_1, S_2, S_3, ..., S_m\}$ yang merupakan subset dari *U*. Akan dicari subset $S' \subseteq S$ dengan kardinalitas minimum sehingga semua elemen pada *U* terdapat pada subset *S'* atau $\cup_{S_j \in S'} S_j = U$.

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.

Skema algoritma *greedy-set-cover* tanpa bobot adalah sebagai berikut,

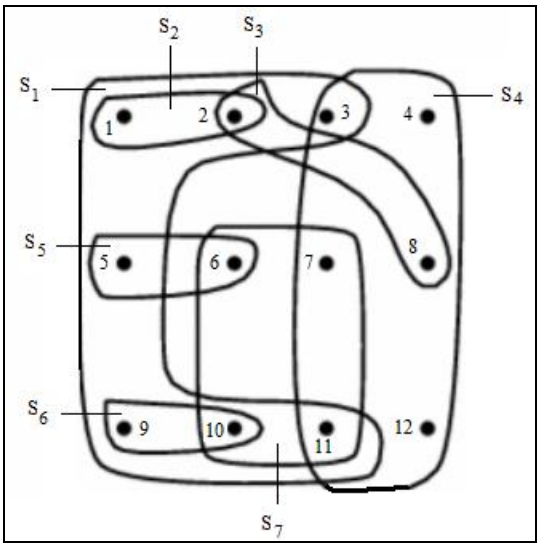
```
function greedy-set-cover (input X : h_elemen, F : h_subset) → h_subset

Deklarasi
   U : h_elemen
   C : h_subset
   S : subset

Algoritma
   U ← X
   C ← {}
   while (U ≠ {}) do
      S ← SELEKSI(F){pilih 1 S∈F yang memaksimalkan |S∩U| }
      U ← U - S
      C ← C ∪ {S}
   endwhile
   return C
```

Masukan dari fungsi ini adalah *X* yang merupakan himpunan semesta elemen dan *F* yang merupakan himpunan subset dari *X*. Keluarannya adalah himpunan subset *C* yang menutupi semua elemen pada *X*. Pada

algoritma ini, *U* adalah himpunan elemen yang belum ditutup, *C* adalah himpunan subset terpilih yang akan menjadi solusi, dan *S* adalah subset yang dipilih pada setiap langkah.

Pada setiap langkah algoritma *greedy-set-cover*, akan dipilih subset *S* dari himpunan subset *F*. Subset yang dipilih tersebut adalah subset yang memaksimalkan $|S \cap U|$ 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 *S* ditambahkan menjadi anggota himpunan subset *C*. Langkah ini terus diulangi sampai semua elemen pada *X* ditutupi atau *U* adalah himpunan kosong.

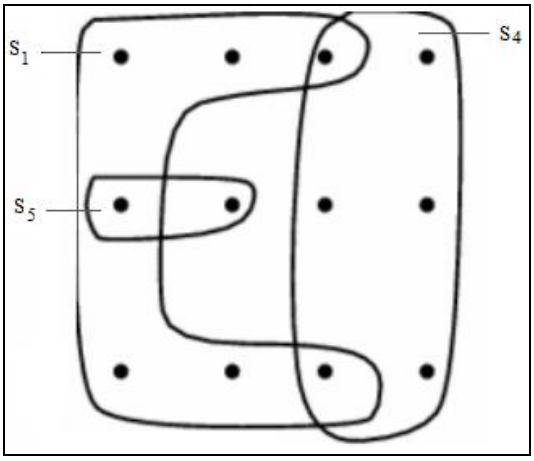
Berikut ini adalah contoh penyelesaian sebuah *set cover problem* dengan algoritma *greedy-set-cover*.



Gambar 2. Contoh *set cover problem* yang yang akan diselesaikan dengan algoritma *greedy-set-cover*

Pada *set cover problem* tersebut *X* = {1,2,3,4,5,6,7,8,9, 10,11,12} dan $F = \{S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$. Pada langkah pertama, dipilih S_1 dipilih karena $|S \cap U| = 7$ dengan $S \cap U$ = {1, 2, 3, 5, 6, 9, 10, 11} merupakan subset yang mampu menutupi elemen yang belum ditutup dengan jumlah paling banyak. Kemudian, himpunan elemen *U* dikurangi dengan elemen yang ada pada subset S_1 dan S_1 ditambahkan pada himpunan solusi *C*.

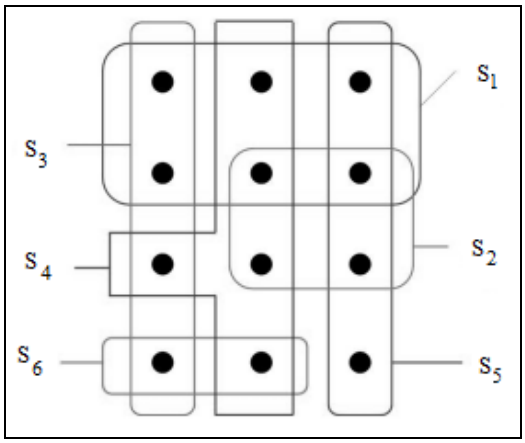
Pada langkah selanjutnya, berturut-turut dipilih S_4 dan S_5. Dengan demikian, himpunan solusi yang dihasilkan adalah $C = \{S_1, S_4, S_5\}$. Solusi tersebut merupakan solusi optimum dari *set cover problem* di atas.



Gambar 3. Solusi *set cover problem* yang diselesaikan dengan algoritma *greedy-set-cover*

2.2 Analisis Algoritma *Greedy-Set-Cover*

Sesuai prinsip algoritma *greedy*, pada setiap langkah algoritma *greedy-set-cover* dipilih subset *S* yang merupakan optimum lokal pada langkah tersebut dengan harapan bahwa langkah-langkah selanjutnya akan mengarah ke solusi optimum global. Namun, pada kenyataannya memang algoritma *greedy* tidak selalu memberikan solusi optimum global. Misalnya pada contoh persoalan *set cover* berikut ini.



Gambar 4. Contoh *set cover problem* yang solusinya jika diselesaikan dengan algoritma *greedy-set-cover* bukan merupakan solusi optimum

Solusi dari *set cover problem* di atas secara berurutan jika diselesaikan dengan algoritma *greedy* adalah $C = \{S_1, S_4, S_5, S_3\}$. Sedangkan solusi optimumnya adalah $C = \{S_3, S_4, S_5\}$.

Dari contoh tersebut, dapat dilihat bahwa algoritma *greedy* tidak selalu memberikan solusi optimum, namun solusi yang dihasilkan *greedy* akan selalu dalam batas antara solusi optimum dan jumlah elemen dalam himpunan semesta.

Hal tersebut dikemukakan dalam teorema berikut ini. Jika solusi optimum mengandung *m* subset, algoritma *greedy* akan menghasilkan solusi *set cover* dengan paling banyak *m* log_e *n* subset.

Misalnya semesta *U* terdiri dari *n* elemen dan dianggap bahwa solusi optimal berukuran *m.* Subset yang pertama dipilih dengan algoritma *greedy* berukuran paling sedikit *n/m*. Dengan demikian, jumlah elemen dalam *U* yang harus ditutupi setelah subset pertama dipilih adalah $n_1 \le n - n/m = n(1-1/m)$.

Sekarang, ada n_1 elemen yang akan ditutup. Paling tidak, sebuah subset yang tersisa mengandung paling sedikit $n_1/(m-1)$ elemen karena jika tidak begitu maka solusi optimumnya akan berukuran lebih dari *m* subset. Setelah algoritma *greedy* memilih subset yang mengandung jumlah elemen yang belum ditutup paling banyak, akan ada $n_2 \le n_1 - n_1/(m-1)$ elemen lagi yang belum ditutup. Dapat kita lihat bahwa $n_2 \le n_1(1-1/(m-1)) \le n_1(1-1/m) \le n_1(1-1/m)^2$. Secara umum diperoleh bahwa,

$$n_{i+1} \le n_i(1-1/m) \le n\ (1-1/m)^{i+1} \qquad (1)$$

Sekarang akan kita tentukan jumlah langkah yang akan dilakukan algoritma *greedy-set-cover* sampai semua elemen pada *U* ditutupi. Jumlah langkah ini akan berhubungan dengan jumlah subset yang akan dipilih oleh algoritma untuk menutupi elemen pada himpunan semesta. Jika dianggap ada *k* langkah untuk menutupi *U*, kita akan memiliki $n_k \le n(1-1/m)^k$ yang nilainya harus kurang dari satu.

$$\begin{aligned} n(1-1/m)^k &< 1 \\ n(1-1/m)^{m\frac{k}{m}} &< 1 \\ (1-1/m)^{m\frac{k}{m}} &< 1/n \\ e^{-\frac{k}{m}} &< 1/n \qquad (1-x)^{\frac{1}{x}} \approx 1/e \\ k/m &> \log_e n \\ k &< m\log_e n \end{aligned} \qquad (2)$$

Dari sini, dapat kita perolah bahwa jumlah subset yang dipilih oleh *greedy-set-cover* akan lebih besar dari *m* log_e *n*. Hal ini membuktikan teorema yang telah disebutkan sebelumnya. Dari hal ini pula, dapat ditunjukkan bahwa *greedy-set-cover* memberikan O($log_e n$) aproksimasi untuk solusi optimum dari *set cover problem* dan pada kenyataannya tidak ada algoritma polinomial yang mampu memberikan aproksimasi yang lebih baik daripada algoritma ini kecuali jika P = NP.

3. KESIMPULAN

Dari penjelasan mengenai penyelesaian *set cover problem* dengan algoritma *greedy* di atas, dapat diperoleh kesimpulan sebagai berikut :

1. Algoritma greedy dapat memecahkan masalah optimasi, namun tidak selalu menghasilkan solusi optimum global. Begitu pula yang terjadi pada algoritma *greedy* yang digunakan untuk menyelesaikan *set cover problem*.
2. Kriteria yang digunakan pada fungsi SELEKSI algoritma *greedy-set-cover* untuk mendapatkan optimum lokal adalah subset yang mampu menutupi jumlah elemen yang belum ditutupi paling banyak.
3. Solusi yang dihasilkan dengan algoritma greedy-set-cover terbukti akan selalu dalam batas antara solusi optimum dan *m* log_e *n*.
4. Algoritma *greedy-set-cover* memberikan O($log_e n$) aproksimasi untuk solusi optimum.
5. Tidak ada algoritma polinomial yang mampu memberikan aproksimasi untuk *set cover problem* yang lebih baik daripada algoritma *greedy-set-cover* kecuali jika P = NP.

REFERENSI

[1] Rinaldi Munir, "Diktat Kuliah IF3051 Strategi Algoritma", Program Studi Teknik Informatika ITB, 2009.

[2] David Hinkemeyer. "Advanced Algorithms", The University of Wisconsin, 2007

[3] Frank Nielsen. "A Concise and Practical Introduction to Programming Algorithms in Java", Springer, 2009

[4] Neal E. Young. "Greedy Set-Cover Algorithms", University of California, 2008

[5] Rohit Prabhavalkar. "Advanced Algorithms", Department of Computer and Information Sciences The Ohio State University, 2009