PENCARIAN *CLIQUE* DALAM GRAF DENGAN MENGGUNAKAN ALGORITMA *BACK-TRACKING*

Adventus Wijaya Lumbantobing

Program Studi Teknik Informatika
Sekolah Teknik Elektro Informatika
Institut Teknologi Bandung
Jl. Ganesha 10, Labtek V, Bandung
e-mail : if15112@students.if.itb.ac.id

ABSTRAK

Permasalahan c*lique* merupakan sebuah permasalahan graf NP-*complete*. Clique adalah sebuah upagraf lengkap yang diperoleh dari suatu graf yang terdiri atas tiga atau lebih simpul. Setiap simpul tersebut saling berdekatan satu sama lain dengan simpul lainnya.

Pencarian maksimum *clique* telah diterapkan dalam sejumlah bidang keilmuan, keahlian teknik dan bahkan dalam bidang bisnis. Beberapa penerapan *clique* meliputi solusi permasalahan struktur molekul DNA. pemrosesan citra dalam pengaturan jarak jauh, penyocokan titik koordinat dalam sistem informasi, permasalahan partisi data dalam kepingan memori dan lain-lainnya.

Permasalahan *clique* ini merupakan permasalahan pencarian solusi yang dianggap cukup sulit, sama seperti permasalahan *knapsack*, pewarnaan graf, sirkuit Hamilton, *n-puzzle* dan TSP. Sampai sekarang ini belum ditemukan algoritma yang benar-benar mangkus untuk penyelesaian permsalahan-permasalahan tersebut meskipun untuk beberapa kasus telah ditmukan algoritma yang memberikan solusi yang mendekati hasil yang optimal.

Pencarian untuk permasalahan *clique* ini dapat dilakukan dengan menggunakan beberapa metode seperti dengan menggunakan algoritma *brute-force* dan dengan penggunaan algoritma *back-tracking* atau runut balik.

Algoritma *back-tracking* digunakan dalam pencarian *clique* yang saling terhubung dalam graf dengan struktur pohon. Pengimplementasian algoritma *back-tracking* menggunakan metode *Depth-First Search* (DFS).

1. PENDAHULUAN

Sebuah *clique* merupakan sebuah upagraf lengkap yang maksimal yang terdiri atas 3 atau lebih simpul yang saling terhubung. Secara matematis pengertian *clique* dapat dituliskan dalam definisi berikut:

> Jika diketahui sebuah graf $G = (V,E)$ dan sebuah bilangan bulat k, sebuah *clique* adalah sebuah subset U dari V dengan $|U|^3k$ sehingga setiap pasangan dari titik verteks dalam U saling terhubung.

Dalam suatu graf akan terdapat sebuah *clique* jika dalam graf tersebut setidaksnya terdapat sebuah upagraf lengkap berjumlah k buah simpul yang dapat berdiri sendiri.

Jumlah mkasimum *clique* yang munkin diperoleh dari suatu graf V dengan ukuran *clique* k dapat dinyatakan secara matematis :

$$\binom{V}{k} = \frac{V!}{k!(V-k)!} \qquad (1)$$

2. METODE

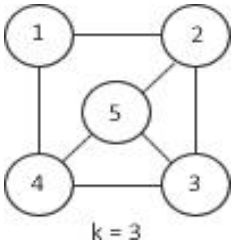
Metode pencarian *clique* dapat menggunakan algoritma-algoritma pencarian solusi seperti *brute-force*, *back-tracking* dan algoritma *union-find* dan algoritma pencarian lainnya. Dalam pencarian *k-clique* (*clique* berukuran k buah simpul) dengan algoritma *brute-force* dilakukan pengurutan set-set dengan nilai k. Untuk setiap simpul dalam tiap set yang bersesuaian dilakukan pemeriksaan jika setiap simpul saling bertetangga atau terhubung. Akan tetapi dalam kasus pencarian *clique*, algoritma ini tidak mangkus dan sulit diimplementasikan untuk permasalahan ini. Sedangkan algoritma *back-tracking* menghasilkan hasil yang lebih mangkus dibandingkan algoritma *brute-force.*

2.1 Algoritma

Algoritma *back-tracking* yang digunakan yaitu dengan menggambarkan graf dalam struktur pohon lalu akar dari pohon dianggap sebagai sebuah *clique* tunggal. Dua buah *clique* akan bergabung jika setiap simpul dalam kedua

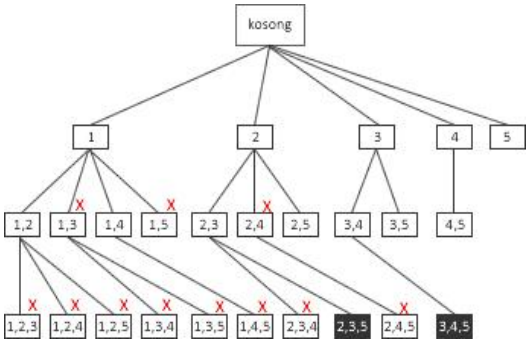
buah *clique* saling terhubung. Pencarian *clique* dalam graf strukutr pohon ini menggunakan algoritma *back-tracking*.

Contoh berikut menunjukkan pencarian *clique* dari graf *G* dengan nilai *k* = 3 menggunakan algoritma *backtracking* :



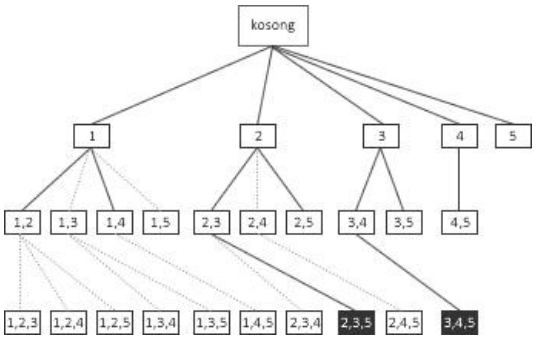
Gambar 1. Graf *G*

a. Algoritma *back-tracking* dengan menghitung semua kemungkinan dari setiap upagraf lalu memeriksa jika simpul tersebut terhubung dengan simpul lainnya dan ukuran *clique* bernilai *k*.



Gambar 2. Metode *back-tracking* I graf *G*

b. Algoritma *back-tracking* yang hanya memeriksa upagraf yang simpul-simpulnya saling terhubung dengan simpul yang lain dan memeriksa ukuran *clique* dengan nilai *k*.



Gambar 3. Metode *back-tracking* II graf *G* (metode *cutoffs pruning*)

Dalam hal efisiensi algoritma *back-tracking* yang kedua dengan metode *cutoffs pruning* lebih mangkus karena dengan algoritma tersebut upagraf yang tidak termasuk solusi tidak diperiksa lebih dalam lagi dengan metode DFS. Dari hasil pencarian dengan algoritma *back-tracking*, dari kedua cara tersebut memberikan hasil *clique* yang sama yaitu upagraf antara simpul 2,3,5 dan antara simpul 3,4,5 dan tiap simpul dari upagraf-upagraf tersebut saling terhubung satu dengan lainnya.

2.2 *Pseudocode*

Berikut ini adalah *pseudocode* untuk pencarian *k-clique* dengan menggunakan *back-tracking* dengan metode *cutoffs pruning* dengan masukan graf sebagai vektor dan sebuah bilangan integer *j*:

```
CliqueBT (input A:vektor, j:integer)
{ jika j sama dengan ukuran clique, k, maka A
  adalah k-clique dari graf }
Algoritma:
  if (j == sizeClique) then
    numClique++
    return
  endif
  else
    j = j + 1
    if (j <= sizeClique) then
     { S adalah semua kemungkinan vektor dari
       j-clique }
      S  = getVektor (A)
       j
    endif
    if (S  tidak kosong) then
        j
    { Untuk setiap kemungkinan dari S ,
                                     j
      lakukan bac-tracking secara rekursif
      untuk k-clique }
      for (untuk setap vektor a  dalam S ) do
                               j         j
         CliqueBT (a , j)
                   j
      endfor
    endif
  endif
```

Pseudocode untuk mengambil daftar vektor dalam S_j sesuai dengan algoritma di atas dituliskan sebagai berikut:

```
getVektor (Vector A)
  daftarVektor { adalah S  , list semua vektor
                         j
                untuk clique }
  { Jika A kosong,  S  adalah vektor pada
                    j
    tiap simpul tunggal dalam graf. }
    if (A kosong) then
      daftarVektor.add (setiap simpul dalam
      graf)
    endif
    else
      { Hitung semua daftar vektor }
      for (tiap elemen j > A.elemenAkhir) do
        isSemuaTerhubung := true
```

```
          { Periksa jika j berdekatan dengan
            Semua verteks dalam A }
          for (tiap tetangga i dari j) do
            if ( i and j tidak terhubung) then
              { Cutoffs pruning }
              isSemuaTerhubung := false
              break
            endif
          endfor
          if (isSemuaTerhubung == true) then
            daftarVektor.add (s )
                              j
          endif
    endfor
  endif
  return daftarVektor
```

Algoritma tersebut di atas bertujuan untuk mencari *clique* dalam suatu graf. Sedangkan untuk mencari *clique* maksimum (atau sering juga disebut *Maximum clique problem*) algoritma umumnya (tanpa menggunakan algoritma *back-tracking*) adalah sebagai berikut:

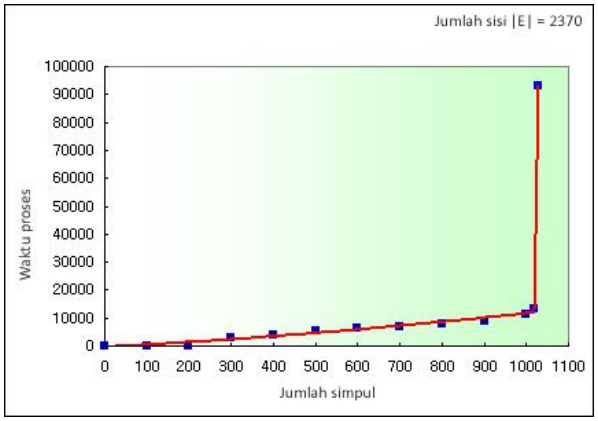
```
maximumClique(input:ccVektor,daftarVektor, I/O:
              k)
  while daftarVektor ≠ ø do
     pilih x dalam daftarVektor lalu buang
     simpan daftarVektor
     tambah x ke ccVektor
     buang simpul y dari daftarvektor
     if filter (ccVektor,daftarVektor,k) then
       if daftarVektor = ø then
          k←ccVektor
       endif
       else
          maksClique(ccVektor,daftarVektor,k)
       endif
     endif
     remove x from Current
    restore Candidate
  endwhile
```

```
filter (input:ccVektor,daftarVektor, I/O: k)
  do
    lanjut ← false
    for setiap y dalam daftarVektor do
      if |Γ(y)∩daftarVektor|+|ccVektor|<|k|
      then
        buang y dari daftarVektor
        if |daftarVektor|+|ccVektor|<|k| then
          lanjut←true
        endif
      endif
    endfor
  while lanjut
  return true
```

3. ANALISIS

Kemangkusan algoritma *back-tracking* untuk pencarian clique dalam graf dapat diperiksa dengan melakukan pengujian. Pengujian dilakukan dengan membandingkan waktu kerja atau waktu yang diperlukan algoritma untuk mencari solusi dengan jumlah simpul dari graf atau dengan jumlah sisi dari graf.

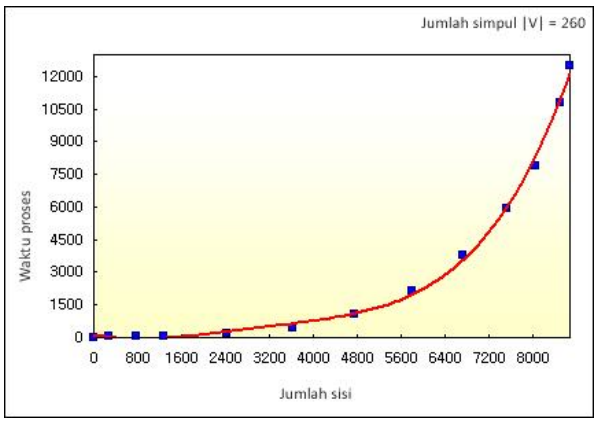
a. Hasil perbandingan antara waktu pemrosesan dengan jumlah simpul (jumlah sisi ditetapkan sebagai konstanta) :



Gambar 4. Perbandingan waktu dengan jumlah simpul

Dalam pengujian yang pertama jumlah sisi ditetapkan sejumlah 2370. Dari grafik di atas, untuk jumlah simpul dari 0 sampai ±1020, fungsi hampir mengikuti bentuk fungsi polinom. Akan tetapi setelah 1030 fungsi tersebut melonjak drastis mengikuti fungsi eksponensial.
Pada kenyataannya fungsi yang dibentuk memang fungsi eksponensial waktu yang bergantung pada nilai *k*. Oleh karena itu dapat diperkirakan titik-titik kritis dari fungsi tersebut adalah 1030, 4100, dan seterusnya, sesuai fungsi eksponensial. Dengan demikian, berdasarkan fungsi tersebut kasus terburuk memiliki kompleksitas $\boldsymbol{O(n^k)}$ dengan k = $\boldsymbol{O(\log n)}$.

b. Hasil perbandingan antara waktu pemrosesan dengan jumlah sisi (jumlah simpul ditetapkan sebagai konstanta) :



Gambar 5. Perbandingan waktu dengan jumlah sisi

Dalam pengujian yang kedua jumlah simpul ditetapkan seniali 260. Karena jumlah simpul ditetapkan senilai 260 makanilai dari *k* juga tatap seniali 4. Dan dengan demikian, kasus terburuk dari kasus ini adalah $O(n^4)$.

Walaupun kemangkusan *k-clique* dengan algoritma *back-tracking* bergantung pada struktur dari graf itu sendiri, algoritma tersebut masih memiliki kompleksitas waktu untuk kasus terburuk $O(n^k)$ dengan nilai *k* sebagai variabel ukuran dari *clique*. Nilai *n* dapat berupa nilai dari simpul atau sisi yang masing-masing dapat dinyatakan dalam suatu fungsi yang saling berlawanan. Fungsi yang dihasilkan merupakan fungsi eksponensial.

4. KESIMPULAN

Pencarian *clique* dari suatu graf dapat dilakukan dengan metode back-tracking. Metode yang digunakan adalah dengan menggambarkan graf dalam bentuk pohon untuk mempermudah pencarian *back-tracking* dengan metode DFS. Algoritma ini akan memeriksa hubungan antar simpul dalam suatu upagraf dengan upagraf lainnya sekaligus memeriksa ukuran upagraf (nilai *k*).

Akan tetapi, algoritma ini bukanlah algoritma yang benar-benar mangkus dalam pengiimplementasiannya dalam permasalahan ini. Masih diperlukan algoritma yang lebih baik yang dapat menghasilkan hasil yang optimal untuk pencarian *clique* dalam suatu graf, misalnya dengan menggunakan algoritma *union-find*.

5. REFERENSI

[1] Alon, Noga, dkk,*Finding a large hidden clique in a random graph.*Tel Aviv University, 1998.

[2] Munir, Rinaldi, *Diktat Kuliah IF2251 Strategi Algoritmik*, ITB, 2007.

[3] Regin, Charles dan Regin,*Using Constraint Programming to Solve the Maximum Clique Problem*,ILOG Sophia Antipolis.

[4] http://en.wikipedia.org/wiki/Clique_problem, Tanggal 19 Mei 2007, Pukul 22.30 WIB.

[5] http://en.wikipedia.org/w/index.php?title=Computers_and Intractability:_A_Guide_to_the_Theory_of_NP-Completness, , Tanggal 19 Mei 2007, Pukul 22.40 WIB.

[6] http://www.ibluemojo.com/school/clique_algorithm.html, Tanggal 19 Mei 2007, Pukul 22.10 WIB.