PENCARIAN *CLIQUE* DALAM GRAF

Adventus Wijaya Lumbantobing

Program Studi Teknik Informatika Institut Teknologi Bandung
Jalan Ganesha 10 Bandung
e-mail: if15112@students.if.itb.ac.id

ABSTRAK

Subgraf meupakan bagian dari graf yang setiap simpulnya saling terhubung satu sma lain.

Clique* adalah sebuah subgraf yang terdapat dalam suatu graf. Permasalahan c*lique* merupakan sebuah permasalahan graf NP-*complete.

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.

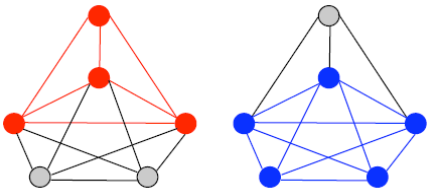
Permasalahan *clique* ini merupakan permasalahan pencarian solusi yang dianggap cukup sulit, sama seperti permasalahan *knapsack*, pewarnaan graf, sirkuit Hamilton, *n-puzzle* dan TSP. Sampai sekarang ini belum ditemukan algoritma yang benar-benar mangkus untuk penyelesaian permsalahan-permasalahan tersebut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

Pencarian clique dapat dilakukan dengan berbagai metode seperti bruteforce, backtracking bond-kerbosch dan berbagai algoritma lainnya.

Kata kunci: *clique, bruteforce, backtracking, bond-kerbosch*

1. PENDAHULUAN

Clique merupakan sebuah upagraf dalam suatu graf yang setiap simpul-simpulnya saling berdekatan dan terhubung. Sebuah clique memiliki paling tidak tiga buah simpul. *Clique* maksimum adalah clique dengan subgraf clique terbesar yang terdapat dalam graf.



Gambar 1 Contoh *clique* dalam graf

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

Jika diketahui sebuah graf *G = (V,E)* dan sebuah bilangan bulat *k*, sebuah *clique* adalah sebuah subset *C* dari *V* sehingga setiap pasangan dari titik verteks dalam *C* saling dihubungkan oleh sebuah sisi.

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

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

2. METODE

Clique dapat dicari dengan menggunakan algoritma-algoritma *brute-force*, *back-tracking* dan algoritma *Bond-Kerbosch* dan algoritma pencarian lainnya.

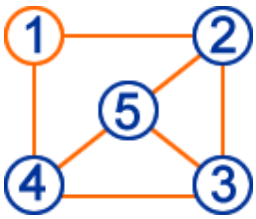
2.1 Algoritma *Brute Force*

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.

Dalam kasus pencarian *clique*, algoritma ini tidak mangkus dan sulit diimplementasikan untuk permasalahan ini.

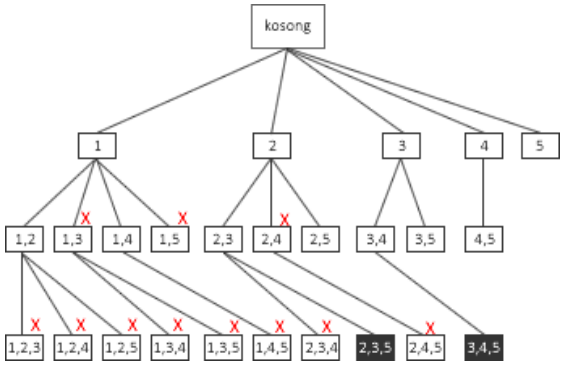
2.2 Algoritma *Backtracking*

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.Berikut adalah contoh yang menunjukkan pencarian *clique* dari graf *G* dengan *k* =3 :



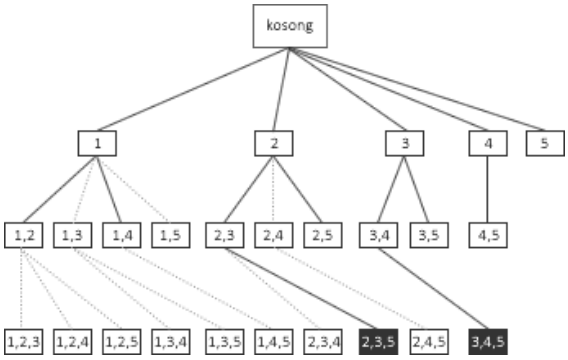
Gambar 2 Graf *G* dengan k=3

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 3 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 4 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.

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_j , list semua vektor
                 untuk clique }
  { Jika A kosong,  S_j adalah vektor pada
    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
```

2.3 Algoritma *Bond-Kerbosch*

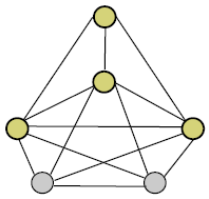
Mencari semua *clique* dalam graf membutuhkan biaya yang besar. Jumlah *clique* dapat bertumbuh secara eksponensial sesuai dengan pertambahan simpul pada graf.

Algoritma *Bond-Kerbosch* dapat menemukan semua clique dalam waktu linear dan merupakan salah satu algoritma yang tercepat dalam pencarian *clique* saat ini.

1. Simpul yang termasuk dalam suatu *clique* dalam graf dilambangkan dengan
2. Kandidat, yaitu simpul yang terhubung dengan subgraf
3. NOT, simpul yang sudah valid yang merupakan bagian dari subgraf dan tidak diakses lagi
4. Kandidat yang dipilih
5. Simpul yang tidak dimasukkan dalam pencarian

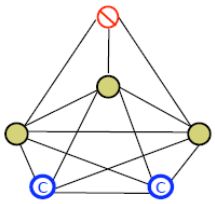
Berikut adalah skema algoritma Bond-Kerbosch dalam contoh graf pada Gambar 1:

1. Inisialisasi.
   Suatu clique ditemukan jika dan hanya jika tidak ada kandidat lagi dan tidak ada simpul dalam himpunan NOT jika tidak, maka clique tidak maksimal



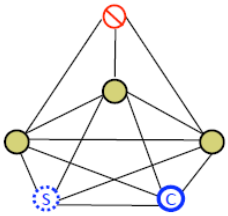
Gambar 5 Clique pertama dalam graf

2. Prosedur rekursif.
   a. Masukan salah satu simpul kedalam set NOT sehingga terdapat tiga simpul dalam subgraf dan terdapat dua kandidat tersisa



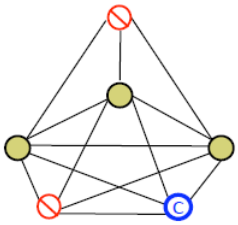
Gambar 6 Pembangkitan simpul kandidat

b. Pilih sebuah kandidat, dan tambahkan ke himpunan subgraf



Gambar 7 Pemilihan kandidat

c. Pilih kandidat baru dan himpunan NOT untuk rekursif berikutnya dengan memasukkan simpul yang tidak terhubung dengan kandidat



Gambar 8 Rekursif pemilihan kandidat baru

3. Terminasi.
   Jika tidak ada kandidat lagi atau terdapat simpul dalam himpunan NOT yang terhubung dengan semua simpul subgraf.

3. HASIL IMPLEMENTASI

Algoritma bruteforce dinilai kurang mangkus dalam menyelesaikan persoalan ini karena sangat membutuhkan biaya dan waktu yang besar.

Algoritma backtracking memberikan hasil yang cukup baik. Kemangkusan *k-clique* dengan algoritma *back-tracking* bergantung pada struktur dari graf itu sendiri, algoritma tersebut masih memiliki kompleksitas waktu untuk kasus terburuk $\boldsymbol{O}(\boldsymbol{n}^{k})$ 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.

Algoritma Bond-Kerbosch merupakan algoritma yang dinilai tercepat saat ini dan sangat efektif karena dapat menemukan smua clique dalam suatu graf.

4. KESIMPULAN

Pencarian *clique* dari suatu graf dapat dilakukan dengan metode bruteforce, back-tracking, Bond-Kerbosch dan berbagai metode lainnya.

Metode bruteforce dilakukan dengan pemeriksaan semua simpul yang terhubung dengan suatu subgraf. Metode backtracking menggunakan graf dalam bentuk pohon untuk mempermudah pencarian 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.

Algoritma Bond-Kerbosch dapat menemukan semua clique dalam suatu graf tak berarah.

REFERENSI

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

[2] Michaela Regneri, dkk, *Finding All Cliques of an Undirected Graph.*2007.

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

[4] *http://en.wikipedia.org/wiki/Clique_(graph_theory)*. Tanggal 4 Januari 2010, Pukul 15.30 WIB.

[5] *http://en.wikipedia.org/wiki/Clique_problem.* Tanggal 4 Januari 2010, Pukul 15.40 WIB.