PENGGUNAAN
ALGORITMA KUHN MUNKRES
UNTUK MENDAPATKAN MATCHING MAKSIMAL
PADA GRAF BIPARTIT BERBOBOT

oleh
GURITNA NOOR AINATMAJA
M0101033

SKRIPSI
ditulis dan diajukan untuk memenuhi sebagian persyaratan
memperoleh gelar Sarjana Sains Matematika

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS SEBELAS MARET
SURAKARTA
2009

SKRIPSI

PENGGUNAAN
ALGORITMA KUHN MUNKRES
UNTUK MENDAPATKAN *MATCHING* MAKSIMAL
PADA GRAF BIPARTIT BERBOBOT

yang disiapkan dan disusun oleh

GURITNA NOOR AINATMAJA

M0101033

dibimbing oleh

Pembimbing I,	Pembimbing II,
Dra. Diari Indriati, M.Si	Dra. Yuliana Susanti, M.Si
NIP 131 805 431	NIP 131 695 845

Telah dipertahankan di depan Dewan Penguji

pada hari Rabu, tanggal 3 Juni 2009

dan dinyatakan telah memenuhi syarat.

Anggota Tim Penguji	Tanda Tangan
1. Drs. Tri Atmodjo K., M.Sc., Ph.D	
NIP 131 791 750	1. .....
2. Dra. Mania Roswitha, M.Si	
NIP 131 285 863	2. .....
3. Drs. Isnandar Slamet, M.Sc	
NIP 132 000 008	3. .....

Surakarta, 3 Juni 2009

Disahkan oleh

Fakultas Matematika dan Ilmu Pengetahuan Alam

Dekan,	Ketua Jurusan Matematika,
Prof. Drs. Sutarno, M.Sc., Ph.D	Drs. Kartiko, M.Si
NIP 131 649 948	NIP 131 569 203

ABSTRAK

Guritna Noor Ainatmaja, 2009. PENGGUNAAN ALGORITMA KUHN MUNKRES UNTUK MENDAPATKAN *MATCHING* MAKSIMAL PADA GRAF BIPARTIT BERBOBOT, Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Sebelas Maret.

Masalah penempatan calon pegawai ke dalam posisi jabatan pekerjaan dapat dibawa ke dalam graf teori dengan mencari *matching* maksimal pada graf bipartit berbobot. *Matching* maksimal pada graf bipartit berbobot dapat diselesaikan dengan menggunakan algoritma Kun Munkres.

Tujuan dari penulisan ini adalah mendapatkan *matching* maksimal pada graf bipartit berbobot, menentukan kompleksitas *running time* algoritma Kuhn Munkres, dan menyusun sebuah program untuk mencari *matching* maksimal pada graf bipartit berbobot. Metode yang digunakan dalam penulisan skripsi ini adalah studi literatur. Oleh karena itu, materi bersumber dari buku-buku referensi dan jurnal yang berhubungan dengan *matching* maksimal pada graf bipartit berbobot, algoritma, kompleksitas waktu *O*-Besar, dan bahasa pemrograman dengan menggunakan *software* Matlab 6.1.

Dari pembahasan disimpulkan bahwa *matching* maksimal pada graf bipartit berbobot dapat diselesaikan dengan menggunakan algoritma Kuhn Munkres. Kompleksitas waktu dalam kasus terburuk algoritma Kuhn Munkres adalah sebesar $O(n^4)$. Untuk data *n vertex* yang besar dapat diselesaikan dengan pembuatan program menggunakan *software* Matlab 6.1.

ABSTRACT

Guritna Noor Ainatmaja, 2009. THE APPLICATION OF KUHN MUNKRES ALGORITHM TO FIND MAXIMUM MATCHING ON WEIGHTED BIPARTITE GRAPH, Faculty of Mathematics and Natural Sciences Sebelas Maret University.

Assignment problem of aplicants for several jobs can be carried out to graph theory by finding maximum matching in weighted bipartite graph. Maximum matching in weighted bipartite graph can be solved using Kuhn Munkres Algorithm.

The purposes of this research are to find maximum matching in weighted bipartite graph, to find complexity of running time Kuhn Munkres Algorithm, and to create a program for finding maximum matching in weighted bipartite graph. The method used in this research is a literary study. Therefore, the references of this research are taken from some books and journals which discuss about maximum matching in weighted bipartite graph, algorithm, Big-*O* time complexity and programming language using Matlab 6.1.

The result shows that maximum matching in weighted bipartite graph can be solved using Kuhn Munkres Algorithm. The complexity of running time in the worst case on this process produces an $O(n^4)$. For *n* vertices it can be solved by a program using Matlab 6.1 software.

MOTO

- **Berusaha sekuat tenaga, dan berdoa-lah,**
- **Jangan berputus asa karena sesungguhnya tidaklah ada yang berputus asa dari rahmat Allah kecuali orang-orang yang kafir (*Q. S. Yusuf : 87*).**

PERSEMBAHAN

Tulisan ini dipersembahkan kepada :

1. Bapak dan Ibu yang tercinta,
2. kedua adikku tersayang.

KATA PENGANTAR

Puji syukur kehadirat Allah SWT yang telah melimpahkan rahmat dan hidayah-Nya, sehingga penulis dapat menyelesaikan skripsi ini.

Penulis menyadari bahwa selesainya skripsi ini tidak lepas dari bimbingan, petunjuk, saran, dan dukungan dari berbagai pihak. Untuk itu penulis mengucapkan terima kasih kepada :

1. Ibu Dra. Diari Indriati, M.Si sebagai Dosen Pembimbing Akademis dan juga selaku Dosen Pembimbing I yang telah meluangkan waktu untuk memberikan bimbingan, nasehat dan pengarahan dalam penyusunan skripsi ini.
2. Ibu Dra. Yuliana Susanti, M.Si sebagai Dosen Pembimbing II yang telah memberikan bantuan dan pengarahan serta perhatian dalam penulisan skripsi ini.
3. Teman-teman angkatan 2001 atas bantuan, semangat, serta dukungan untuk menyelesaikan skripsi ini.

Penulis berharap tulisan ini dapat menambah wawasan mahasiswa FMIPA UNS, terutama tentang teori graf.

Surakarta, Juni 2009

Penulis

DAFTAR ISI

Halaman

DAFTAR GAMBAR

Halaman

DAFTAR LAMPIRAN

Halaman

NOTASI

E	: Himpunan *edge*
e	: Nama *edge* suatu graf
f, g	: fungsi pada bilangan Asli, N
G	: Nama suatu graf
H	: Nama suatu subgraf
i, j, p, q, k	: Indeks
K	: Nama suatu graf lengkap
ℓ	: Nama suatu pelabelan *vertex*
M	: Nama suatu *matching*
N	: Himpunan bilangan Asli
O	: Notasi O-Besar (Big-O)
P	: Nama suatu lintasan
Q	: Nama suatu barisan memuat *vertex* tingkatan genap pada pohon *Alternating*
r	: Nama *vertex* akar suatu graf pohon
T	: Graf yang berbentuk pohon
U, V	: Himpunan *vertex* suatu graf
u, v	: Nama *vertex* suatu graf
w	: Bobot suatu *edge*
$\deg_G$	: Notasi *degree* suatu *vertex*
Δ	: Notasi *degree* terbesar suatu graf
δ	: Notasi *degree* terkecil suatu graf

มหาวิทยาลัยราชภัฏรำไพพรรณี

BAB I
PENDAHULUAN

1.1 Latar Belakang Masalah

Sebuah perusahaan membuka lowongan pekerjaan dengan syarat yang telah ditentukan. Seorang pelamar diberi kesempatan untuk memilih lebih dari satu jabatan. Perusahaan melakukan ujian seleksi untuk mendapatkan calon pegawai dengan nilai terbaik untuk setiap jabatan pekerjaan. Perusahaan ingin mengoptimalkan calon pekerja dengan jabatan pekerjaan yang tersedia. Jika dilakukan penggolongan antara calon pegawai dengan posisi jabatan menjadi dua himpunan yang berhubungan, dan hasil nilai ujian dijadikan sebagai relasi penghubung antara calon pegawai dengan jabatannya maka masalah ini dapat dibawa ke dalam graf *matching* bipartit berbobot. Himpunan calon pegawai U dan posisi jabatan V sebagai himpunan *vertex* yang dihubungkan dengan nilai hasil ujian sebagai bobot *edge* $w(U,V)$. Pengoptimalan penempatan calon pegawai ke jabatan dapat dikaji dengan menggunakan teori graf, yaitu dengan cara mendapatkan *matching* maksimal pada graf bipartit berbobot. *Matching* maksimal pada suatu graf berbobot G adalah *matching* yang memiliki jumlah bobot yang maksimal.

Matching maksimal pada graf bipartit berbobot dapat diperoleh dengan menggunakan algoritma Kuhn Munkres. Rosen (2003) menyatakan bahwa keefisienan suatu algoritma dapat ditentukan atas dasar dari jumlah waktu dan memori yang dibutuhkan untuk menyelesaikan suatu *input* dengan ukuran tertentu. Jumlah waktu dan memori yang dibutuhkan algoritma ini dapat diukur dengan kompleksitas yang dinotasikan dengan O-Besar.

1.2 Perumusan Masalah

Berdasarkan latar belakang masalah di atas, maka permasalahannya dapat dirumuskan sebagai berikut :

1. bagaimana mendapatkan *matching* maksimal pada graf bipartit berbobot,
2. berapa besar keefisienan algoritma Kuhn Munkres,
3. bagaimana menyusun sebuah program untuk mendapatkan *matching* maksimal pada graf bipartit berbobot.

1.3 Batasan Masalah

Dalam penulisan skripsi ini, permasalahan dibatasi pada penentuan *matching* dalam graf bipartit berbobot, penghitungan kompleksitas *running time* suatu algoritma menggunakan *O*-Besar kompleksitas waktu terburuk, aplikasi menggunakan *software* Matlab 6.1.

1.4 Tujuan

Tujuan dari penulisan ini adalah

1. mendapatkan *matching* maksimal pada graf bipartit berbobot,
2. menentukan kompleksitas *running time* algoritma Kuhn Munkres,
3. menyusun sebuah program untuk mencari *matching* maksimal pada graf bipartit berbobot.

1.5 Manfaat

Manfaat dari penulisan makalah ini adalah :

1. menambah pengetahuan tentang graf teori dan aplikasinya,
2. mendapatkan hasil yang maksimal dari penempatan calon pegawai ke jabatan yang sesuai dengan keahliannya.

BAB II
LANDASAN TEORI

Bab ini dibagi menjadi dua bagian yaitu tinjauan pustaka dan kerangka pemikiran. Pada bagian tinjauan pustaka diberikan beberapa terminologi tentang graf yang digunakan dalam penulisan ini. Pada bagian kerangka pemikiran dijelaskan alur dalam penulisan skripsi ini.

2.1 Tinjauan Pustaka

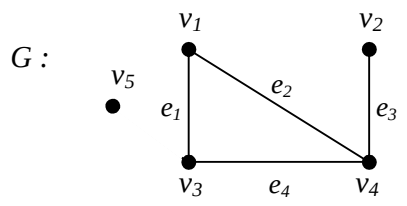
Untuk mencapai tujuan penulisan, diperlukan teori-teori yang mendukung dan relevan dengan pembahasan. Pada subbab ini diberikan beberapa definisi, teorema dan pengertian yang berhubungan dengan *matching graph bipartite* berbobot, meliputi pengertian graf, graf terhubung, graf *tree*, graf *bipartite*, *matching*, dan algoritma Kuhn Munkres.

2.1.1 Definisi dan Notasi Graf

Definisi 2.1. (Chartrand dan Lesniak, 1986:4) *Suatu graf G adalah pasangan himpunan* (V,E), *dimana V adalah himpunan vertex berhingga yang tidak kosong dan E adalah himpunan edge yang menghubungkan sepasang vertex di dalam V. Graf G dapat dinyatakan dengan* $G=(V,E)$. *Jika* $V=\{v_1,v_2,v_3,...,v_n\}$ *dan e adalah edge yang menghubungkan vertex* v_i *dan* v_j, *maka e dapat dinyatakan dengan* $e=(v_i,v_j)$.

Definisi 2.2. (Chartrand dan Oellerman, 1993:3) *Banyaknya vertex dalam himpunan V disebut order, sedang banyaknya edge dalam himpunan E disebut size.*

Gambar 2.1 merupakan graf *G*, dengan *order* 5 dan *size* 4.



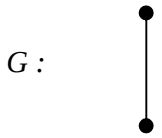
Gambar 2.1. Graf *G*

Definisi 2.3. (Bondy dan Murty, 1976:3) *Vertex v_1 dan v_2 dalam G dikatakan adjacent (bertetangga) jika keduanya dihubungkan oleh sebuah edge. Suatu vertex dikatakan incident jika vertex tersebut dikawankan dengan edge. Sedangkan dua edge dikatakan adjacent jika dua edge tersebut incident pada sebuah vertex yang sama.*

Pada Gambar 2.1 terlihat *vertex v_1* dan v_3 dikatakan *adjacent, vertex v_1 incident* dengan *edge e_1*, sedang *edge e_1* dan *e_2 adjacent.*

Definisi 2.4. (Chartrand dan Oellerman, 1993:6) *Degree dari vertex v dalam graf G adalah banyaknya edge yang incident dengan v, yang dinotasikan* $\deg_G v$. *Suatu graf G disebut r-reguler atau reguler degree r, jika setiap vertex dari graf G mempunyai degree yang sama sebesar r. Degree terbesar dari suatu vertex dalam graf G disebut maximum degree* $\Delta(G)$ *dan degree terkecil dari suatu vertex dalam graf G disebut minimum degree* $\delta(G)$. *Vertex yang mempunyai degree 0 disebut isolated vertex (vertex terasing), sedang vertex yang mempunyai degree 1 disebut end-vertex.*

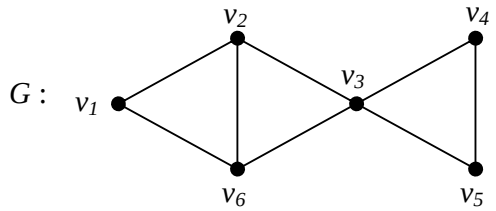
Gambar 2.1 deg v_2 = 1, deg v_5 = 0, $\Delta(G)$ = 3, $\delta(G)$ = 0, v_5 adalah *isolated vertex,* sedang v_2 adalah *end-vertex*. Contoh graf *1-reguler* graf terlihat pada Gambar 2.2 dibawah ini.



Gambar 2.2. *1-reguler graph*

Definisi 2.5. (Chartrand dan Lesniak, 1986:26) *u dan v adalah vertex dari graf G, walk dari u ke v dalam graf G adalah rangkaian bergantian dari vertex dan edge dari G, dimulai dari u berakhir v. Trail dalam graf G adalah walk dimana tidak mengulang edge. Lintasan (path) dari u ke v adalah walk u-v dimana tidak mengulang vertex. Suatu vertex u dapat membentuk lintasan trivial u. Circuit dalam suatu graf G adalah suatu trail dimana vertex awal sama dengan vertex akhir, definisi suatu cycle adalah circuit yang tidak mengulang vertex kecuali pada vertex awal dan akhir dengan jumlah edge paling sedikit tiga edge.*

Gambar 2.3 contoh *walk* $v_3 - v_4$ adalah v_3, v_2, v_6, v_3, v_4, v_5, v_4. Contoh *trail* adalah v_3 , v_2 , v_6 , v_3 , v_4 sedang suatu *path* $v_1 - v_5$ adalah v_1 , v_2 ,v_3 , v_4 , v_5. Contoh sebuah *circuit* adalah v_2 , v_3 , v_4 , v_5 , v_3 , v_6 , v_2 sedang v_1, v_2 , v_3 , v_6 , v_1 adalah sebuah *cycle*.

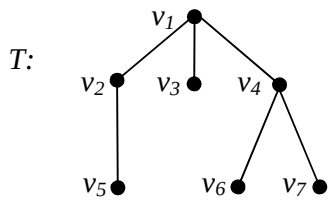


Gambar 2.3. *Walk, Path, Trail, Circuit dan Cycle* dalam graf *G*.

Definisi 2.6. (Chartrand dan Lesniak, 1986:28) *Graf G dikatakan terhubung* (*connected*) *jika untuk setiap dua vertex berbeda u dan v, terdapat suatu lintasan u-v yang menghubungkannya.*

Definisi 2.7. (Chartrand dan Lesniak, 1986:26) *Graf terhubung yang tidak memuat suatu cycle disebut pohon (tree). Sebuah graf pohon T disebut pohon berakar jika T mempunyai satu vertex akar yaitu r dimana untuk setiap vertex v di T terdapat lintasan r ke v.*

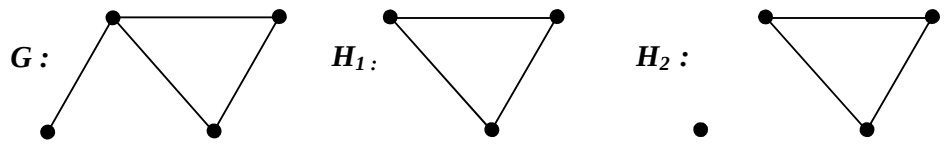
Contoh pohon terlihat pada Gambar 2.4.



Gambar 2.4. Pohon *T*

Definisi 2.8 (Bondy dan Murty, 1976:8) *Graf H disebut subgraf dari graf G jika* $V(H) \subseteq V(G)$ *dan* $E(H) \subseteq E(G)$. *Graf H disebut subgraf perentang* (*spanning subgraph*) *G jika order subgraf H sama seperti order graf G,* $V(H) = V(G)$ *dan* $E(H) \subseteq E(G)$.

Contoh subgraf H_1 dan subgraf perentang H_2 dapat dilihat pada gambar di bawah ini.

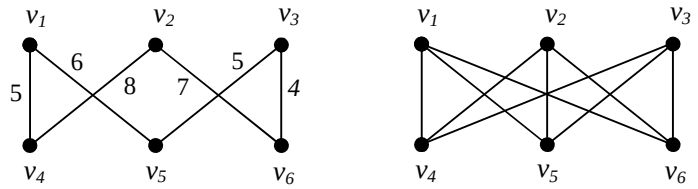


Gambar 2.5. H_1 subgraf G dan H_2 adalah subgraf perentang G

Definisi 2.9 (Bondy dan Murty, 1976:5) *Graf G dikatakan graf bipartit jika vertex dari G, V(G) dapat dipartisi menjadi dua sub himpunan tidak kosong* V_1 *dan* V_2 *dimana edge dari G incident dengan vertex* V_1 *dan vertex* V_2. *Graf bipartit lengkap adalah graf bipartit dimana setiap vertex dari* V_1 *adjacent dengan semua vertex dari* V_2. *Jika* $|V_1| = m$ *dan* $|V_2| = n$, *maka graf bipartit lengkap dinotasikan dengan* $K_{m,n}$. *Jika setiap edge dari graf G diberi nilai atau bobot, maka disebut graf berbobot.*

Gambar 2.6 (*a*) menunjukkan graf bipartit berbobot dengan *vertex V(G)* dipartisi menjadi sub himpunan $V_1 = \{v_1, v_2, v_3\}$ dan $V_2 = \{v_4, v_5, v_6\}$ serta tiap *edge* menghubungkan *vertex* dari V_1 dan V_2, dengan diberi nilai atau bobot. Gambar 2.6 (*b*) merupakan graf bipartit lengkap.

.



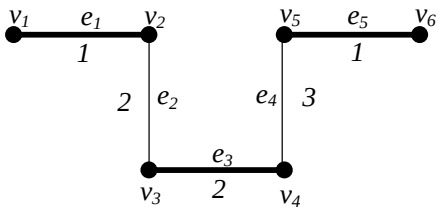
(*a.*) (*b.*)

Gambar 2.6. Graf Bipartit

2.1.2 *Matching* Graf Bipartit

Definisi 2.10. (Chartrand dan Oellermann, 1993:162) *Matching dalam graf G adalah 1-reguler subgraf dari G, merupakan subgraf yang diambil dari himpunan pasangan edge yang tidak adjacent, dengan notasi M. Edge dari G yang termasuk dalam M disebut edge matching, sedang edge dari G yang bukan termasuk dalam matching disebut bukan edge matching. Sebuah vertex v dalam G adalah vertex matching jika v incident dengan edge dalam M, jika tidak maka v adalah vertex tunggal dalam M.*

Gambar 2.7 menunjukkan *matching* $M_1 = \{e_2, e_4\}$, dimana e_2 dan e_4 merupakan *edge matching* dari M_1, sedang e_1, e_3, e_5 bukan *edge matching* dari M_1. *Vertex matching* M_1 adalah v_2, v_3, v_4, v_5 sedang v_1 dan v_6 adalah *vertex* tunggal.



Gambar 2.7. *Matching*

Definisi 2.11. (Toroslu dan Ucoluk, 2006) *Lintasan alternating dari G adalah lintasan yang memiliki edge yang saling bergantian antara edge matching dan bukan matching. Lintasan alternating yang dimulai dan diakhiri dengan titik tunggal disebut lintasan augmenting.*

Graf *G* pada Gambar 2.7, dimana *matching* $M_1 = \{e_2, e_4\}$, terdapat lintasan v_1, v_2, v_3, v_4, v_5 yang merupakan lintasan *alternating* terhadap M_1, tapi bukan lintasan *augmenting,* sedang lintasan $v_1, v_2, v_3, v_4, v_5, v_6$ adalah lintasan *augmenting* terhadap M_1.

Definisi 2.12. (Chartrand dan Oellermann, 1993:162) *Matching yang memiliki anggota yang maksimal dalam graf G disebut matching maksimal dari G.*

Matching maksimal pada graf berbobot adalah matching yang memiliki jumlah bobot edge yang maksimal. Graf G adalah graf berorder p yang mempunyai matching dengan jumlah p/2 edge maka matching tersebut adalah matching sempurna.

Matching $M_2 = \{e_1, e_3, e_5\}$ pada Gambar 2.7 adalah *matching* maksimal dan juga *matching* sempurna, sedangkan $M_1 = \{e_2, e_4\}$ adalah *matching* maksimal berbobot.

2.1.3 Algoritma

Algoritma, menurut Fletcher *et al.* (1991:123) adalah daftar instruksi yang linear, digabung dengan *input* tertentu yang memiliki dua sifat sebagai berikut :

1. setiap *input* menghasilkan suatu *output* dalam waktu yang berhingga,
2. algoritma bersifat seperti fungsi, yaitu satu *input* tidak mungkin menghasilkan dua *output* yang berbeda.

Seperti yang ditulis Chartrand dan Oellerman (1993:171), untuk mendapatkan *matching* maksimal pada graf bipartit dapat diperoleh dengan menggunakan algoritma *matching* maksimal. Langkah-langkah dalam algoritma sebagai berikut.

2.1. Algoritma *Matching* Maksimal untuk Graf Bipartit

Misal *G* adalah graf bipartit dengan $V_1(G) = \{v_1, v_2, ..., v_p\}$ dan $V_2(G) = \{y_1, y_2, ..., y_q\}$ dengan *matching* awal M_1, maka langkah untuk mendapatkan *matching* maksimal pada graf bipartit sebagai berikut :

1. Indeks variabel *i* menandai tahapan di algoritma yang berakar pada v_i dari pohon *alternating*. Himpunan *M* akan memuat *matching* maksimal di penghentian algoritma. Langkah awal *i* adalah 1 dan *M* memuat *edge* dari M_1.

   $i \leftarrow 1$ dan $M \leftarrow M_1$.

2. Langkah kedua menentukan apakah *matching M* adalah *matching* maksimal. Jika $i < p$, maka lanjutkan, dan jika tidak, maka berhenti karena *matching* maksimal *M* sudah didapatkan.
3. Langkah ketiga mencari *vertex* tunggal di *matching M* yang sebelumnya belum digunakan sebagai akar dari pohon *alternating*. Jika *vertex v* didapatkan, maka *v* menjadi akar dari pohon *alternating* baru. Langkah ini awal sebuah barisan *Q*, yaitu barisan yang memuat *v* saja. Barisan *Q* memuat *vertex* pada tingkatan genap dari pohon *alternating* dan *vertex* yang bertetangga tetap diuji. Jika v_i adalah *vertex matching*, maka $i \leftarrow i+1$ dan kembali ke Langkah 2, dan jika tidak, maka $v \leftarrow v_i$ dan *Q* diawali memuat *v* saja.
4. Langkah keempat membangun pohon *alternating* yang berakar pada *v*.
   a. Untuk $j = 1,2,..., p$ dan $j \neq i$, misal pohon$(v_j) \leftarrow F$ dan pohon $(v_i) \leftarrow T$.

      Langkah ini awal penyusunan pohon, yaitu susunan yang memuat tepat satu entri untuk setiap *vertex* dari *G*. Susunan pohon digunakan untuk menandai adanya *vertex* dari pohon *alternating* yang sedang dibangun. Jika *vertex* sudah termasuk pohon *alternating*, maka pohon(*v*) = *T* (*True*), dan jika *vertex* belum termasuk pohon *alternating*, maka pohon(*v*) = *F* (*False*).
   b. Jika $Q = \phi$, maka $i \leftarrow i+1$ dan kembali ke Langkah kedua, dan jika tidak, maka hapus titik *x* dari *Q* dan lanjutkan. Langkah ini menentukan apakah susunan pohon *alternating* berakar pada $v = v_i$ adalah lengkap.
   c. Selanjutnya menguji setiap *vertex* yang bertetangga dengan *x* untuk menentukan apakah pohon *alternating* dapat diperluas atau lintasan *augmenting* sudah didapatkan. Jika ada *edge* dari *vertex x* ke *y*, maka *x* disebut *parent y*.

      1). Misal $N(x) = \{y_1, y_2,..., y_k\}$ dan $j \leftarrow 1$.

2). Jika $j \leq k$, maka $y \leftarrow y_j$ dan jika tidak, maka kembali ke Langkah b.

3). Jika pohon(y) = T, maka $j \leftarrow j+1$ dan kembali ke Langkah 2). dan jika tidak, maka lanjutkan.

4). Jika y menempel dengan *edge matching yz*, maka y adalah titik tunggal dan lanjutkan.

5). Susunan *parent* digunakan untuk menentukan lintasan *alternating v* ke x yaitu P' di pohon *alternating*. Jika P adalah lintasan *augmenting* yang diperbolehkan dari P' dengan menambah lintasan x ke y, maka lakukan Langkah 5.

5. Jika *matching* baru M' diperoleh dari *augmenting* M sepanjang P, maka algoritma berhenti. Jika $M \leftarrow M'$ dan $i \leftarrow i+1$, maka kembali ke Langkah 2. Langkah ini *augmenting* M sepanjang P dan mengganti M dengan *matching* baru yang sudah dibentuk.

2.1.4 Kompleksitas Waktu Algoritma

Menurut Chartrand dan Oellerman (1993:38), kompleksitas suatu algoritma diukur dengan seberapa besar usaha yang dikeluarkan dari hasil komputerisasi ketika komputer menyelesaikan masalah menggunakan algoritma tersebut. Ukuran ini bisa berdasarkan jumlah langkah komputerisasi, *running time*, atau seberapa besar ruang yang diperlukan untuk menyimpan.

Fletcher *et. al.* (1991:134) menyatakan dalam pembuatan suatu program ada dua hal yang penting untuk dijadikan bahan pertimbangan selain dari kebenaran program tersebut, yaitu waktu yang diperlukan untuk mengeksekusi dan besar *memory* yang dibutuhkan dalam komputer. Seiring perkembangan teknologi sekarang, terbatasnya besar *memory* bukan menjadi pertimbangan utama lagi, yang menjadi pertimbangan utama dalam pembuatan program adalah waktu yang diperlukan untuk memproses inputan data yang masuk dan menghasilkan output.

Waktu yang diperlukan suatu algoritma sangat bergantung pada ukuran *input* yang diproses. Bahkan dapat dipilih suatu *input* berukuran tertentu yang jika dimasukkan ke dalam algoritma akan membutuhkan waktu yang terbaik (tercepat) untuk menghasilkan *output*. Selain itu, juga dapat dipilih *input* yang berukuran lain yang membutuhkan waktu terburuk (terlama) untuk menghasilkan suatu *output.* Kasus semacam ini sering dikenal dengan kasus terbaik dan kasus terburuk. Selain itu juga dikenal kasus rata-rata, yaitu waktu rata-rata untuk semua ukuran *input.* Pada penelitian ini dikaji untuk kasus waktu terburuk.

Waktu yang diperlukan suatu algoritma dalam menyelesaikan suatu *input* akan bertambah seiring bertambahnya ukuran *input*. Besaran yang digunakan untuk menentukan waktu terburuk yang dibutuhkan suatu algoritma adalah kompleksitas yang dinotasikan dengan *O*-Besar. Definisi *O*-Besar menurut Johnsonbaugh (1991: 43) adalah sebagai berikut.

Definisi 2.13. *Misal fungsi f dan g adalah fungsi yang bekerja pada domain bilangan asli,* N. *Kemudian hubungan antara f dan g dituliskan*

$$f(n) = O(g(n))$$

dan dikatakan f(n) berorder paling besar g(n) jika terdapat konstan positif $c > 0$ *sedemikian hingga*

$$|f(n)| \leq c|g(n)|$$

untuk semua bilangan bulat positif yang berhingga n.

Pernyataan " $f(n)$ berorder paling besar $g(n)$" memberikan pengertian bahwa jika suatu algoritma memiliki kompleksitas waktu sebesar $O(g(n))$ dan *n* diperbesar, maka waktu yang dibutuhkannya tidak akan melebihi suatu konstanta *c* dikalikan dengan $g(n)$.

Rosen (2003) menyatakan bahwa notasi *O*-Besar menyajikan batas atas waktu yang dibutuhkan algoritma untuk menyelesaikan *input* berukuran *n* dalam kasus terburuk. Lebih lanjut, waktu sebenarnya yang dibutuhkan algoritma dapat

diketahui jika setiap langkah di dalam algoritma telah diterjemahkan ke dalam suatu bahasa pemrograman yang kemudian dijalankan oleh komputer.

2.2 Kerangka Pemikiran

Sebuah perusahaan membuka lowongan pekerjaan dengan syarat yang telah ditentukan. Seorang pelamar diberi kesempatan untuk memilih lebih dari satu jabatan. Perusahaan melakukan ujian seleksi untuk mendapatkan calon pegawai dengan nilai terbaik untuk setiap jabatan pekerjaan. Berpijak pada landasan teori di atas, masalah pengoptimalan penempatan calon pekerja ke dalam posisi jabatan dapat diselesaikan dengan mencari *matching* maksimal pada graf bipartit berbobot menggunakan algoritma Kuhn Munkres.

Selanjutnya, mencari keefisienan algoritma Kuhn Munkres dalam menyelesaikan *matching* maksimal pada graf bipartit berbobot dengan mengetahui kompleksitas waktu *O*-Besar.

Untuk *input* data yang besar akan lebih mudah mendapatkan penyelesaian dengan membuat suatu program. Program dibuat dengan menggunakan *software* Matlab 6.1.

BAB III
METODE PENELITIAN

Metode yang digunakan dalam penulisan skripsi ini adalah studi literatur, sehingga langkah-langkah yang diambil adalah mengumpulkan beberapa buku dan sumber pustaka lain yang berkaitan dengan *matching*, *matching* graf bipartit, *matching* graf bipartit berbobot, algoritma, kompleksitas *O*-Besar dan pemrograman Matlab 6.1. Untuk mencapai tujuan skripsi ini, maka diambil langkah-langkah sebagai berikut :

1. Diberikan pengertian dasar tentang graf, lintasan, graf pohon, graf bipartit, *matching*, algoritma, *matching* pada graf bipartit berbobot.
2. Mengkaji langkah-langkah algoritma Kuhn Munkres.

   Sesuai dengan judul tulisan ini, perlu diberikan penjelasan langkah-langkah algoritma Kuhn Munkres menghasilkan *matching* maksimal pada graf bipartit berbobot.
3. Mencari kompleksitas algoritma Kuhn Munkres.

   Untuk mencari keefisienan algoritma Kuhn Munkres dapat ditunjukkan dengan mencari kompleksitas dalam menyelesaikan masalah *matching* maksimal pada graf bipartit berbobot.
4. Menerapkan algoritma Kuhn Munkres dalam bahasa pemrograman.

   Untuk mempermudah mencari *matching* maksimal pada graf bipartit berbobot dibuat program menggunakan algoritma Kuhn Munkres dengan software Matlab 6.1.
5. Implementasi algoritma Kuhn Munkres.

   Diberikan suatu graf bipartit berbobot dicari *matching* maksimal dengan menggunakan algoritma Kuhn Munkres secara manual maupun dengan program, serta membandingkan hasil keduanya.

BAB IV
PEMBAHASAN

Pemahaman dasar yang cukup diperlukan untuk mempermudah memahami suatu masalah. Beberapa teorema berikut merupakan dasar dalam penyusunan algoritma untuk mendapatkan *matching* maksimal pada graf bipartit berbobot.

4.1. *Matching*

Teorema 4.1 Misal M_1 dan M_2 adalah *matching* di graf G. Jika H merupakan subgraf perentang dari G dengan himpunan *edge* $E(H) = (M_1 - M_2) \cup (M_2 - M_1)$, maka setiap komponen dari H mengikuti salah satu dari tipe berikut :

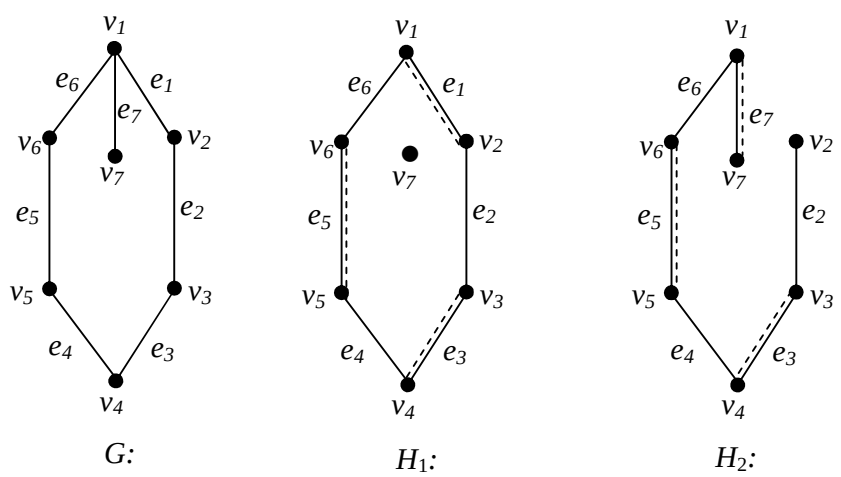
a. Sebuah *vertex* terasing
b. Sebuah *cycle* genap dengan *edge* saling bergantian antara M_1 dan M_2
c. Sebuah lintasan tak *trivial* dengan *edge* saling bergantian antara M_1 dan M_2 sedemikian sehingga setiap *vertex* terakhir dari lintasan adalah *vertex* tunggal di M_1 atau M_2 tetapi bukan keduanya.

Bukti: $\Delta(H) \leq 2$ karena setiap *vertex* dari H *incident* dengan paling banyak satu *edge* di M_1 dan M_2. Akibatnya setiap komponen dari H adalah sebuah lintasan (mungkin *trivial*) atau sebuah *cycle*. Jika komponen dari H adalah lintasan *trivial*, maka komponen tersebut dinamakan *vertex* terasing. Karena tidak ada dua *edge* yang *adjacent* pada *matching*, maka *edge* dari setiap *cycle* dan lintasan di H adalah saling bergantian antara M_1 dan M_2. Jadi setiap *cycle* di H adalah genap.

Misal $e = uv$ adalah *edge* di H, dan u adalah *vertex* terakhir di lintasan P yang juga komponen dari H. Akan ditunjukkan bahwa u adalah *vertex* tunggal di M_1 atau M_2, tapi bukan keduanya. Karena $e \in E(H)$, maka berlaku $e \in M_1 - M_2$ atau $e \in M_2 - M_1$. Jika $e \in M_1 - M_2$ dan u adalah *vertex matching* di M_1, ditunjukkan u adalah *vertex* tunggal di M_2. Andaikan u bukan *vertex* tunggal di M_2. Misal *edge* f di M_2 (dengan $f \neq e$) sedemikian hingga f *incident* dengan u

(f dan e *adjacent*). Karena e dan f *adjacent*, maka $f \notin M_1$. Jadi $f \in M_2 - M_1 \subseteq E(H)$. Hal ini tidak mungkin karena u adalah *vertex* terakhir di P. Jadi u adalah *vertex* tunggal di M_2. Dengan cara yang sama, dapat ditunjukkan bahwa jika $e \in M_2 - M_1$, dan u adalah *vertex matching* di M_2, maka u adalah *vertex* tunggal di M_1. ❑

Gambar 4.1 menunjukkan H (H_1 dan H_2) merupakan subgraf perentang dari G yang mempunyai himpunan *edge*, $E(H) = (M_1 - M_2) \cup (M_2 - M_1)$ dengan $M_1=\{e_2, e_4, e_6\}$ dan $M_2=\{ e_1, e_3, e_5 \}$ di H_1, $M_1=\{e_2, e_4, e_6\}$ dan $M_2=\{e_3, e_5, e_7\}$ di H_2. Graf H_1 terdiri dari 2 komponen, yaitu *vertex* terasing v_7 dan *cycle* dengan panjang 6, sedangkan graf H_2 terdiri dari 1 komponen, yaitu lintasan yang *edge*-nya saling bergantian antara M_1 dan M_2.



Gambar 4.1. H_1 dan H_2 merupakan subgraf perentang dari G

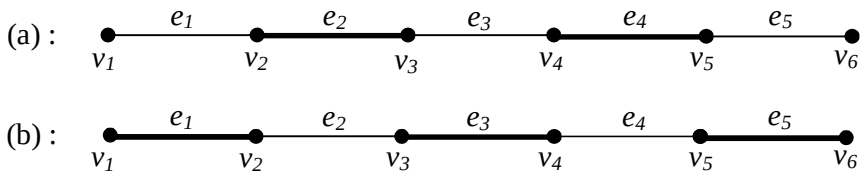
Teorema 4.2 Sebuah *matching* M di graf G adalah *matching* maksimal jika dan hanya jika tidak ada lintasan *augmenting-M* di G.

Bukti : Misal M adalah *matching* maksimal di G. Andaikan G memuat P, yaitu lintasan *augmenting-M*, maka P mempunyai panjang ganjil. Jika M' adalah himpunan *edge* dari M yang ada di P dan $M''=E(P)-M'$ sehingga $E(M'')>E(M')$, maka $|M''|=|M'|+1$ dan himpunan $M_1=(M-M')\cup M''$ adalah *matching* dengan $|M_1|=|M|+1$. Jadi M_1 diperoleh dari *augmenting* M

sepanjang P. Ini kontradiksi dengan M di graf G adalah *matching* maksimal, maka tidak ada lintasan *augmenting-M* di G.

Sebaliknya, misal M_1 adalah *matching* di graf G dan tidak ada lintasan *augmenting-*M_1 di G. Akan ditunjukkan bahwa M_1 adalah *matching* maksimal di G. Misal M_2 adalah *matching* maksimal di G. Dari pembuktian bagian pertama tadi, maka tidak ada lintasan *augmenting-*M_2 di G. Misal H adalah subgraf perentang dari G dengan $E(H) = (M_1 - M_2) \cup (M_2 - M_1)$. Misal subgraf H' merupakan komponen dari H. Jika H' adalah *vertex* terasing, maka H' tidak mempunyai *matching* maksimal. Jika H' adalah *cycle* genap, maka $|M_1|=|M_2|$. Dengan demikian M_1 adalah *matching* maksimal di G. Jika H' bukan *vertex* terasing maupun *cycle* genap, maka berdasarkan Teorema 4.1, H' adalah lintasan tak *trivial* dengan *edge* saling bergantian antara M_1 dan M_2. Karena G tidak memuat lintasan *augmenting-*M_1, maka H' adalah lintasan yang mempunyai panjang genap sedemikian hingga $|M_1 - M_2| = |M_2 - M_1|$, yang berarti $|M_1| = |M_2|$. Jadi M_1 adalah *matching* maksimal di G. ❑

Sebagai ilustrasi dari Teorema 4.2, perhatikan graf G pada Gambar 4.2 (a). Misalkan $M = \{e_2, e_4\}$ adalah *matching* di G dan P: v_1,v_2,v_3,v_4,v_5,v_6 adalah lintasan *augmenting-M*. Maka $M_1 = \{e_1,e_3,e_5\}$ adalah *matching* yang diperoleh dengan *augmenting M* sepanjang P, dengan $|M_1| = 3 = |M| + 1$ (lihat Gambar 4.2 (b)).



Gambar 4.2. *Augmenting M* sepanjang P

4.2. *Matching* Maksimal pada Graf Bipartit

Diketahui graf G dengan partisi himpunan *vertex* U_1 dan U_2 di $V(G)$. Misal terdapat *matching* M di G sedemikian hingga setiap *edge* dari M *incident* dengan *vertex* di U_1 dan U_2, dan setiap *vertex* dari U_1 atau U_2 *incident* dengan *edge* di M. Jika $M \subseteq M^*$ dengan M^* adalah *matching* di G, maka *matching* M adalah

matching bagian dari M^* di G.

Persekitaran *vertex* v di graf G dinotasikan $N(v)$ adalah *vertex-vertex* yang bertetangga dengan *vertex* v, yaitu : $N(v)=\{u\in V(G) \mid vu\in E(G)\}$ dan $deg(v) = |N(v)|$. Teorema 4.2 dan hasil berikut merupakan dasar dari Algoritma 4.1 untuk mendapatkan *matching* maksimal pada graf bipartit.

Teorema 4.3 M adalah *matching* di graf G yang bukan maksimal. Misal M_1 adalah *matching* yang diperoleh dengan *augmenting* M sepanjang lintasan *augmenting*, maka G memuat lintasan *augmenting-M* yang diakhiri *vertex* tunggal.

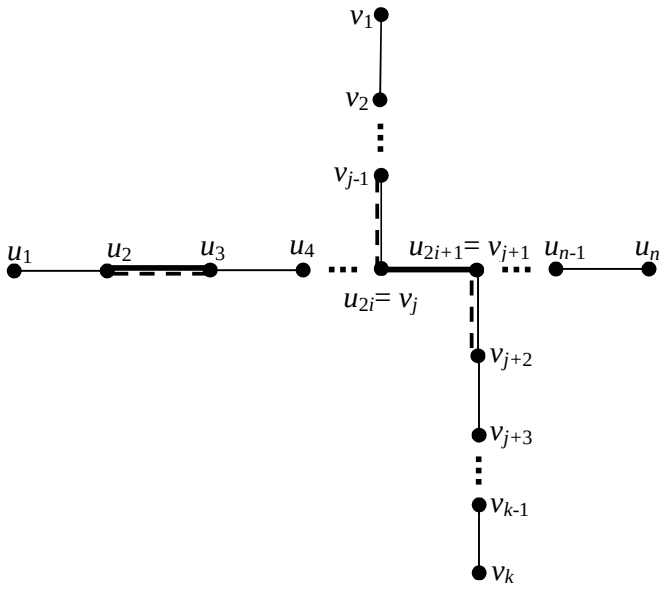
Bukti : Andaikan G tidak memuat lintasan *augmenting-M* yang diakhiri *vertex* tunggal. Misal $P:v=u_1, u_2, ..., u_n$ adalah lintasan *augmenting-*M_1 tetapi bukan lintasan *augmenting-M*, maka P memuat semua *edge* dari M_1. Misal i adalah bilangan asli terkecil sedemikian hingga $u_{2i}u_{2i+1} \in M_1 - M$, maka u_{2i} bukan *vertex* tunggal di M.

Misal M_1 diperoleh dari *augmenting* M sepanjang lintasan $Q:v_1, v_2, ..., v_k$. Jika $u_{2i}u_{2i+1} \in E(Q) - M$, maka u_{2i} *incident* dengan *edge* e di Q dan u_{2i} adalah *vertex matching* di M. Misal $u_{2i} = v_j$ $(1<j<k)$, maka $e = v_{j-1}v_j$ atau $e = v_jv_{j+1}$.

Jika $e = v_{j-1}v_j$, maka lintasan $Q'=v_1, v_2, ..., v_{j-1}v_j$ adalah lintasan *alternating-M* yang memuat tepat satu *vertex* tunggal, yaitu v_1. *Edge* yang bukan *edge matching* di M dari Q' merupakan *edge matching* di M_1 ketika augmenting M sepanjang Q. Jadi Q' dan P' mempunyai *vertex* $v_j=u_{2i}$, sehingga $v=u_1,u_2, ..., u_{2i}, v_{j-1}, v_{j-2}, ..., v_1$ adalah lintasan a*ugmenting-M* dan tidak memuat lintasan *augmenting-*M_1 yang diakhiri *vertex* tunggal (lihat Gambar 4.3). Asumsi ini kontradiksi dengan G memuat lintasan *augmenting-*M_1 yang diakhiri *vertex* tunggal.

Jika $e = v_jv_{j+1}$, maka dengan cara yang sama $v = u_1,u_2, ..., u_{2i}, v_{j+1}, ..., v_k$ adalah lintasan *augmenting-M*. Ini juga menghasilkan kontradiksi. Jadi G tidak memuat lintasan augmenting-M_1 yang diakhiri *vertex* tunggal. ❑

Sebagai ilustrasi, Gambar 4.3 menunjukkan pembuktian Teorema 4.3 menghasilkan lintasan *augmenting-M*. *Matching* M_1 adalah *edge* yang ditandai dengan garis tebal dan *matching M* adalah *edge* yang ditandai dengan garis putus-putus.



Gambar 4.3. Lintasan *Augmenting-M*

Akibat Teorema 4.3 Misal M_1 adalah *matching* awal di graf G. Misal $M_1, M_2, \ldots, M_k$ adalah barisan *matching* di *G* dengan $M_i (2 \le i \le k)$ diperoleh dengan *augmenting* M_{i-1} sepanjang lintasan *augmenting-*M_{i-1}, dan M_k adalah *vertex* tunggal, maka *G* tidak memuat lintasan *augmenting-*M_1 yang diawali *vertex* tunggal, maka *G* tidak memuat lintasan *augmenting-*M_1 $(2 \le i \le k)$ yang diakhiri *vertex* tunggal.

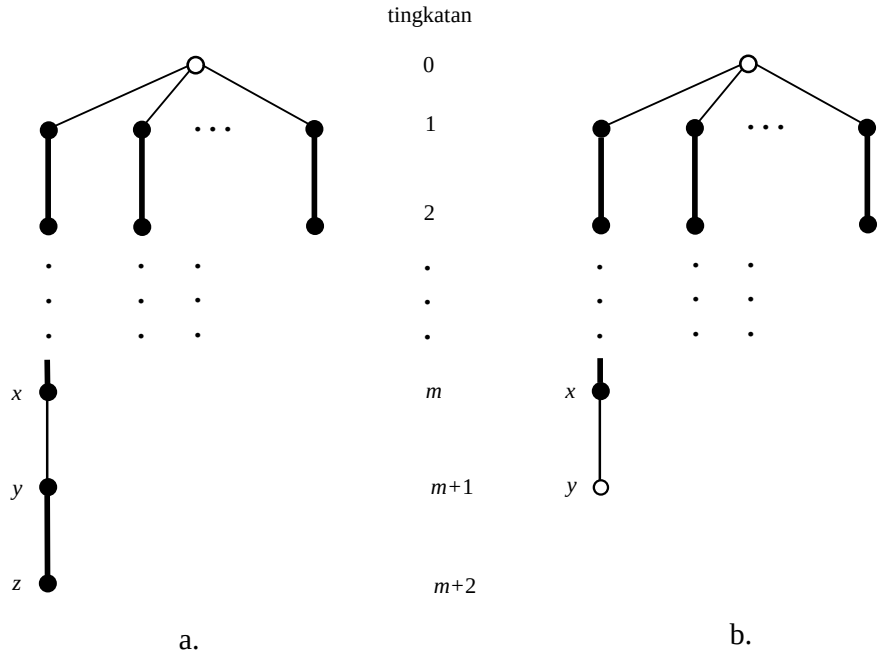
Bukti : Andaikan *G* memuat lintasan *augmenting-*M_1 yang diakhiri *vertex* tunggal, maka berdasarkan Teorema 4.3, *G* memuat lintasan *augmenting-*M_{i-1} yang diakhiri *vertex* tunggal. Misal $i=2$ sehingga $M_{i-1} = M_1$. Jadi *G* memuat lintasan *augmenting-*M_1 yang diawali dan diakhiri oleh *vertex* tunggal. Ini kontradiksi dengan *G* tidak memuat lintasan *augmenting-*M_1 yang diakhiri *vertex* tunggal. ❑

Sebuah cara yang sistematis dan efisien dilakukan untuk mendapatkan *matching* maksimal di graf G. Misal M_1 adalah *matching* awal di graf G. Jika ada lintasan *augmenting*-M_1, maka *augmenting* M_1 sepanjang lintasan *augmenting*-M_1 adalah untuk mendapatkan *matching* baru M_2 dengan $|M_2| = |M_1| + 1$. Jika tidak ada lintasan *augmenting*-M_1 yang diawali *vertex* tunggal, maka M_1 adalah *matching* maksimal.

Pohon alternating merupakan pohon berakar di *vertex* v, jika setiap lintasan yang berawal di v adalah lintasan *alternating*-M. Jika G adalah graf bipartit dan v adalah *vertex* tunggal di *matching* M, maka pohon *alternating* dapat dibangun. Jika ada lintasan *augmenting*-M yang diakhiri *vertex* tunggal, maka lintasan akan diperoleh dari susunan pohon *alternating*. Lintasan *augmenting*-M didapat melalui proses berikut.

Pohon *alternating* yang berakar pada v dibuat dengan penempatan v pada tingkatan 0, dan semua *vertex* $u_1, u_2, ..., u_k$ yang bertetangga dengan v di G ditempatkan pada tingkatan 1, dan *vertex* v dihubungkan ke *vertex* u_i ($1 \leq i \leq k$). Misal u_1v_i untuk $1 \leq i \leq k$. *Vertex* $v_1, v_2, ..., v_k$ ditempatkan pada tingkatan 2, dan u_i dihubungkan ke v_i ($1 \leq i \leq k$) sedemikian hingga tingkatan kedua dari pohon alternating sudah tersusun. Misal pohon *alternating* sudah tersusun sampai tingkatan m (m genap). Pada susunan pohon *alternating*, setiap *vertex* x pada tingkatan m di pohon *alternating* menguji setiap *vertex* y yang bertetangga dengan x. Jika y termasuk pohon *alternating*, maka ada lintasan adalah *augmenting*. Jika y bukan termasuk pohon *alternating*, maka y adalah *vertex matching* atau *vertex* tunggal. Jika y adalah *vertex matching* dengan $yz \in M$, maka z bukan termasuk pohon alternating, sehingga *vertex* y dan z berturut-turut ditempatkan pada tingkatan $m+1$ dan $m+2$ dengan x dihubungkan ke y dan y ke z. Jika y adalah *vertex* tunggal, maka lintasan *augmenting* v ke y yaitu lintasan *alternating* v ke x di pohon *alternating* yang diikuti oleh *edge* xy sudah didapatkan. Gambar 4.4 (a) menunjukkan y adalah *vertex matching* dengan $yz \in M$, sedangkan (b) menunjukkan bahwa y adalah *vertex* tunggal. *Edge matching* ditandai dengan garis tebal dan *vertex matching* ditandai dengan *vertex* gelap.

Susunan pohon *alternating* terselesaikan jika salah satu lintasan terdapat lintasan *augmenting* atau tidak bisa menambah tingkatan pada pohon *alternating*. Pada kasus pertama, *augmenting* M sepanjang lintasan *augmenting* adalah untuk mendapatkan *matching* M_1 dengan jumlah *edge* $|M| + 1$. Jika ada *vertex* tunggal di *matching* baru yang belum digunakan sebagai akar dari pohon *alternating*, maka sebuah *vertex* dipilih untuk menjadi akar dari pohon *alternating* baru. Jika tidak ada *vertex* tunggal, maka M adalah *matching* maksimal.



Gambar 4.4. Membangun Pohon *Alternating*

4.3. *Matching* Maksimal pada Graf Bipartit Berbobot

Misal G adalah graf bipartit berbobot dengan partisi himpunan V_1 dan V_2 dan graf G' adalah graf bipartit lengkap berbobot yang memuat G sebagai subgraf. Misal G' mempunyai partisi himpunan U_1 dan U_2 dengan $|U_1| = |U_2| =$ maksimal $\{|V_1|, |V_2|\}$ dan V_i termuat di U_i, untuk $i = 1,2$. Misal $x \in U_1$ dan $y \in U_2$, jika $w_{G'}(xy) = w_G(xy)$, maka $xy \in E(G)$ dan jika $w_{G'}(xy) = 0$, maka $xy \notin E(G)$. Jika M adalah *matching* maksimal di G', maka M adalah *matching*

sempurna di *G'* dengan beberapa *edge* dari *M* mungkin mempunyai bobot 0, sehingga $M \cap E(G)$ adalah *matching* maksimal di *G*. Jadi jika *M* adalah *matching* sempurna dengan jumlah bobot maksimal di *G'*, maka $M \cap E(G)$ adalah *matching* maksimal di *G*.

Dalam tulisan ini digunakan algoritma Kuhn Munkres, untuk mencari *matching* sempurna dengan jumlah bobot maksimal di *G*. Algoritma Kuhn Munkres mencari *matching* sempurna dengan jumlah bobot maksimal dengan memilih bobot yang terbesar dari persekitaran *vertex v* di V_1 dan membangun sebuah pohon *alternating* yang hasilnya terus diulang sampai mendapatkan sebuah *matching* sempurna dengan jumlah bobot maksimal. Untuk mendapatkan *matching* maksimal di *G* yang *vertex*-nya diberi label, maka diperlukan pelabelan *vertex-vertex* dalam graf *G*.

Pelabelan *vertex* adalah fungsi nilai real ℓ di himpunan *vertex* $V_1 \cup V_2$ untuk semua $v \in V_1$ dan $u \in V_2$,

$$\ell(v) + \ell(u) \geq w(vu) \tag{4.1}$$

Misalkan

$$\ell(v) = \text{maksimal } \{w(vu) \mid u \in V_2\} \text{ untuk semua } v \in V_1$$

dan

$$\ell(u) = 0 \text{ untuk semua } u \in V_2,$$

maka ℓ adalah pelabelan *vertex* di *G'* dengan

$$E_\ell = \{vu\ E(G') \mid v \in V_1 \text{ dan } u \in V_2 \text{ dan } \ell(v) + \ell(u) = w(vu)\}.$$

Teorema 4.4 Misal ℓ adalah pelabelan *vertex* di graf bipartite lengkap berbobot *G'*. Misal H_ℓ adalah subgraf perentang dari *G'* dengan himpunan *edge* E_ℓ. Jika E_ℓ memuat *matching* sempurna *M'*, maka *M'* adalah *matching* maksimal di *G'*.

Bukti : Misal *M'* adalah *matching* sempurna di H_ℓ . Karena H_ℓ adalah subgraf perentang dari *G'*, maka *M'* adalah *matching* sempurna di *G'* dan

$$w(M') = \Sigma\{w(e) \mid e \in M'\} = \Sigma\{\ell(x) \mid x \in V(G')\} \tag{4.2}$$

Misal *M* adalah *matching* sempurna selain *M'* di *G'*, maka menurut (4.1)

$$\Sigma\{\ell(x) \mid x \in V(G')\} \geq w(M) = \Sigma\ \{w(e) \mid e \in M\} \tag{4.3}$$

dan menurut (4.2) dan (4.3), $w(M') \geq w(M)$. Jadi M' adalah *matching* maksimal di G'. ❑

4.4. Langkah Algoritma Kuhn Munkres

Langkah-langkah dalam mendapatkan *matching* maksimal pada graf bipartit berbobot dengan menggunakan algoritma Kuhn Munkres sebagai berikut.

4.4.1. <u>Algoritma Kuhn Munkres</u>

Misal G' adalah graf bipartit berbobot dengan partisi himpunan *vertex* V_1 dan V_2.

1. Langkah awal sebuah pelabelan *vertex* ℓ.
   1.a. Untuk setiap $v \in V_1$, misal $\ell(v) \leftarrow$ maksimal $\{w(vu) \mid u \in V_2\}$.
   1.b. Untuk setiap $u \in V_2$, misal $\ell(u) \leftarrow 0$.
   1.c. Misal H_ℓ adalah subgraf perentang dari G' dengan himpunan *edge* E_ℓ.
   1.d. Misal G_ℓ adalah graf dasar dari H_ℓ.
2. Berlaku Algoritma 2.1 untuk mendapatkan *matching* maksimal M di G_ℓ.
3. Jika mendapatkan *matching* maksimal di G_ℓ, dicari apakah G_ℓ merupakan *matching* sempurna, maka menurut Teorema 4.4 *matching* sempurna tersebut adalah *matching* maksimal di G'.

   Jika *matching* maksimal di G_ℓ bukan *matching* sempurna, maka susun sebuah pohon *alternating* T yang berakar di *vertex* tunggal dengan Algoritma 4.1 untuk mendefinisikan sebuah pelabelan *vertex* baru ℓ'.
   3.a. Jika setiap *vertex* di V_1 merupakan *vertex matching* di M, maka M adalah *matching* maksimal di G' dan algoritma berhenti, jika tidak maka dilanjutkan.
   3.b. Misal x adalah *vertex* tunggal di V_1.
   3.c. Susun pohon *alternating* dari M yang berakar di x. Jika ada lintasan *augmenting-M*, maka *augmenting M* sepanjang lintasan *augmenting-M* untuk mendapatkan *matching* baru dan kembali ke langkah 3.a, jika tidak memuat lintasan *augmenting-M* dan T adalah pohon *alternating*

dari M yang berakar di x yang tidak dapat diperluas lebih jauh di G_ℓ, maka pelabelan *vertex* ℓ diganti dengan sebuah pelabelan *vertex* baru ℓ' dengan sifat bahwa M dan T termuat di graf dasar $G_{\ell'}$ dari $H_{\ell'}$.

4. Langkah berikutnya untuk menghitung pelabelan *vertex* baru ℓ'.

   Misalkan :

   $m_\ell \leftarrow$ minimal $\{\ell(v) + \ell(u) - w(vu) \mid v \in V_1 \cap V(T)$ dan $u \in V_2 - V(T)\}$.

   Maka berlaku untuk ℓ' :

$$\ell'(v) \leftarrow \begin{cases} \ell(v) - m_\ell & \text{untuk} \quad v \in V_1 \cap V(T) \\ \ell(v) + m_\ell & \text{untuk} \quad v \in V_2 \cap V(T) \\ \ell(v) & \text{lainnya} \end{cases}$$

5. Jika $\ell \leftarrow \ell'$, maka buat G_ℓ dan kembali ke Langkah 3.c.

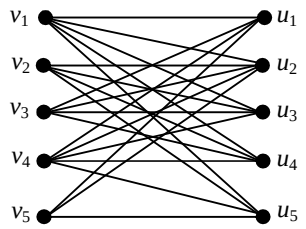
4.4.2. <u>Contoh Kasus</u>

Algoritma Kuhn-Munkres digunakan untuk pengoptimalan penempatan calon pekerja ke jabatan yang sesuai dengan kemampuannya. Sebagai contoh misal terdapat 5 calon pekerjaan, v_i ($i=1,2,\ldots,5$) untuk 5 jabatan dalam suatu perusahaan u_j ($j=1,2,\ldots,5$). Tiap pelamar memiliki kemampuan yang berbeda-beda untuk tiap jabatan. Kemampuan pelamar kerja adalah bobot *edge* yang menghubungkan pelamar kerja v_i dengan jabatan u_j. Bentuk matrik $M = [m_{ij}]$ dengan $m_{ij} = w(v_i u_j)$.

$$M = \begin{array}{c|ccccc} & u_1 & u_2 & u_3 & u_4 & u_5 \\ \hline v_1 & 5 & 1 & 1 & 3 & 2 \\ v_2 & 0 & 1 & 3 & 3 & 4 \\ v_3 & 2 & 5 & 4 & 3 & 0 \\ v_4 & 2 & 2 & 3 & 4 & 4 \\ v_5 & 6 & 2 & 0 & 0 & 1 \end{array}$$

Jika dibuat grafnya sebagai berikut :

G :



G adalah graf bipartit berbobot dengan partisi himpunan *vertex-vertex*, dengan partisi himpunan $V_1 = \{v_1, v_2, v_3, v_4, v_5\}$ dan $V_2 = \{u_1, u_2, u_3, u_4, u_5\}$. Dengan mengikuti langkah-langkah pada algoritma Kuhn Munkres didapatkan hasil sebagai berikut :

Langkah 1.a. Untuk setiap $v \in V_1$, $\ell(v) \leftarrow$ maksimal $\{w(vu) \mid u \in V_2\}$,

sehingga diperoleh : $\ell(v_i) = \{5, 4, 5, 4, 6\}$

1.b. Untuk setiap $u \in V_2$, $\ell(u) \leftarrow 0$, maka nilai pelabelan

$\ell(u_i) = \{0, 0, 0, 0, 0\}$

1.c. H_ℓ adalah subgraf perentang dari G' dengan himpunan *edge* E_ℓ.

1.d. G_ℓ adalah graf dasar dari H_ℓ .

Langkah 2. Berlaku Algoritma 4.1 untuk mendapatkan *matching* maksimal M di G_ℓ , diperoleh $M = \{v_1u_1, v_2u_5, v_3u_2, v_4u_4\}$ adalah *matching* maksimal di G_ℓ.

Langkah 3.a. Ada *vertex* di V_1 merupakan *vertex* tunggal di M, maka lanjutkan.

3.b. v_5 adalah *vertex* tunggal di V_1 sehingga v_5 sebagai *vertex* akar.

3.c. Susun pohon *alternating* dari M yang berakar di v_5. Pohon *alternating* diperoleh dengan $V(T) = \{v_5, u_1, v_1\}$, karena tidak ada lintasan *augmenting-M* dan T adalah pohon *alternating* dari M yang berakar di v_5 yang tidak dapat diperluas lebih jauh di G_ℓ, maka pelabelan *vertex* ℓ diganti dengan sebuah pelabelan *vertex* baru ℓ'.

Langkah 4. Menghitung label baru ℓ' :

$m_\ell \leftarrow$ minimal $\{\ell(v) + \ell(u) - w(vu) \mid v \in V_1 \cap V(T)$ dan $u \in V_2 - V(T)\}$

anggota *vertex* $v \in V_1 \cap V(T)$ adalah $\{v_1, v_5\}$ dan anggota $u \in V_2 - V(T)$ adalah $\{u_2, u_3, u_4, u_5\}$, jadi perhitungan nilai m_ℓ sebagai berikut :

$m_\ell = \ell(v_1) + \ell(u_2) - w(v_1u_2) = 5 + 0 - 1 = 4$

$m_\ell = \ell(v_1) + \ell(u_3) - w(v_1u_3) = 5 + 0 - 1 = 4$

$m_\ell = \ell(v_1) + \ell(u_4) - w(v_1u_4) = 5 + 0 - 3 = 2$

$m_\ell = \ell(v_1) + \ell(u_5) - w(v_1u_5) = 5 + 0 - 2 = 3$

$m_\ell = \ell(v_5) + \ell(u_2) - w(v_5u_2) = 6 + 0 - 2 = 4$

$m_\ell = \ell(v_5) + \ell(u_3) - w(v_5u_3) = 6 + 0 - 0 = 6$

$m_\ell = \ell(v_5) + \ell(u_4) - w(v_5u_4) = 6 + 0 - 0 = 6$

$m_\ell = \ell(v_5) + \ell(u_5) - w(v_5u_5) = 6 + 0 - 1 = 5$

dari perhitungan di atas maka diperoleh $m_\ell = 2$, sehingga diperoleh pelabelan baru ℓ' sebagai berikut :

$$\ell'(v) \leftarrow \begin{cases} \ell(v) - m_\ell = \{3,4,5,4,4\} & \text{untuk} \quad v \in V_1 \cap V(T) \\ \ell(v) + m_\ell = \{2,0,0,0,0\} & \text{untuk} \quad v \in V_2 \cap V(T) \\ \ell(v) & \text{lainnya} \end{cases}$$

Langkah 5. Karena $\ell \leftarrow \ell'$, maka buat G_ℓ dan kembali ke Langkah 3.c.

Langkah 3.c. Susun pohon *alternating* dari M yang berakar di v_5. Pohon *alternating* diperoleh dengan $V(T) = \{v_5, u_1, v_1, u_4, v_4, u_5, v_2\}$, karena tidak ada lintasan *augmenting-M* dan T adalah pohon *alternating* dari M yang berakar di v_5 yang tidak dapat diperluas lebih jauh di G_ℓ, maka pelabelan *vertex* ℓ diganti dengan sebuah pelabelan *vertex* baru ℓ'.

Langkah 4. Menghitung label baru ℓ' :

$m_\ell \leftarrow$ minimal $\{\ell(v) + \ell(u) - w(vu) \mid v \in V_1 \cap V(T)$ dan $u \in V_2 - V(T)\}$

anggota *vertex* $v \in V_1 \cap V(T)$ adalah $\{ v_1, v_2, v_4, v_5 \}$ dan anggota $u \in V_2 - V(T)$ adalah $\{ u_2, u_3 \}$, jadi perhitungan nilai m_ℓ sebagai berikut :

$m_\ell = \ell(v_1) + \ell(u_2) - w(v_1u_2) = 3 + 2 - 1 = 4$

$m_\ell = \ell(v_1) + \ell(u_3) - w(v_1u_3) = 3 + 2 - 1 = 4$

$m_\ell = \ell(v_2) + \ell(u_2) - w(v_2u_2) = 4 + 0 - 1 = 3$

$m_\ell = \ell(v_2) + \ell(u_3) - w(v_2u_3) = 4 + 0 - 3 = 1$

$m_\ell = \ell(v_4) + \ell(u_2) - w(v_4u_2) = 4 + 0 - 2 = 2$

$m_\ell = \ell(v_4) + \ell(u_3) - w(v_4u_3) = 4 + 0 - 3 = 1$

$m_\ell = \ell(v_5) + \ell(u_2) - w(v_5u_2) = 4 + 0 - 2 = 2$

$m_\ell = \ell(v_5) + \ell(u_3) - w(v_5u_3) = 4 + 0 - 0 = 4$

dari perhitungan di atas maka diperoleh $m_\ell = 1$, sehingga diperoleh pelabelan baru ℓ' sebagai berikut :

$$\ell'(v) \leftarrow \begin{cases} \ell(v) - m_\ell = \{2,3,5,3,3\} & \text{untuk} \quad v \in V_1 \cap V(T) \\ \ell(v) + m_\ell = \{3,0,0,1,1\} & \text{untuk} \quad v \in V_2 \cap V(T) \\ \ell(v) & \text{lainnya} \end{cases}$$

Langkah 5. Karena $\ell \leftarrow \ell'$, maka buat G_ℓ dan kembali ke Langkah 3.c.

Langkah 3.c. Susun pohon *alternating* dari M yang berakar di v_5. Pohon *alternating* diperoleh dengan $V(T) = \{v_5, u_1, v_1, u_4, v_4, u_5, v_2, u_3\}$, karena lintasan *augmenting-M* didapatkan, maka *augmenting M* sepanjang lintasan *augmenting-M* untuk mendapatkan *matching* baru dan kembali ke Langkah 3.a.

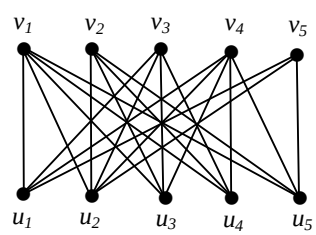
Langkah 3.a. Setiap *vertex* di V_1 merupakan *vertex matching* di M, maka $M=\{v_5u_1, v_1u_4, v_4u_5, v_2u_3, v_3u_2\}$ adalah *matching* maksimal di G' dan algoritma berhenti.

Jadi $M \cap E(G) = \{v_5u_1, v_1u_4, v_4u_5, v_2u_3, v_3u_2\}$ adalah *matching* maksimal di G.

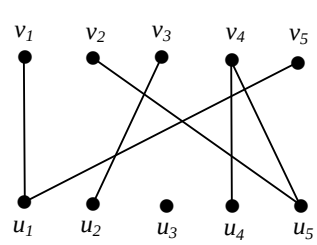
Ilustrasi dalam matriks dan gambar :

G :

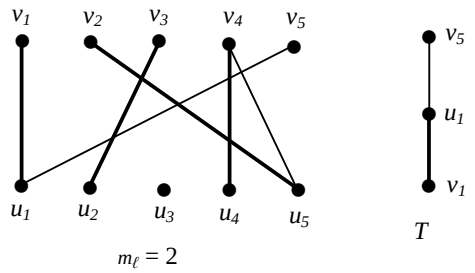
	u_1	u_2	u_3	u_4	u_5
v_1	5	1	1	3	2
v_2	0	1	3	3	4
v_3	2	5	4	3	0
v_4	2	2	3	4	4
v_5	6	2	0	0	1



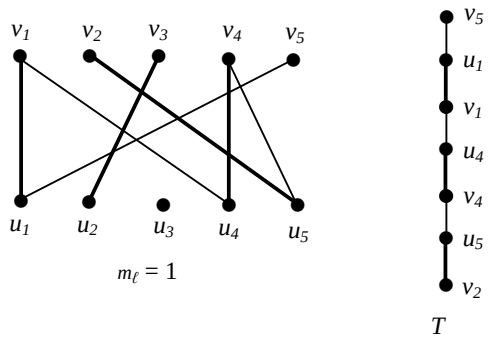
	u_1	u_2	u_3	u_4	u_5	$\ell(v_i)$
v_1	5	1	1	3	2	5
v_2	0	1	3	3	4	4
v_3	2	5	4	3	0	5
v_4	2	2	3	4	4	4
v_5	6	2	0	0	1	6
$\ell(u_i)$	0	0	0	0	0	



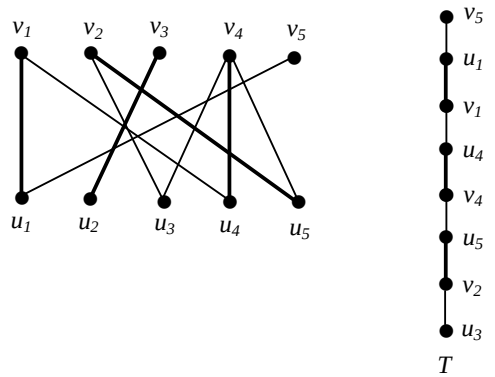
	u_1	u_2	u_3	u_4	u_5	$\ell(v_i)$
v_1	5	1	1	3	2	5
v_2	0	1	3	3	4	4
v_3	2	5	4	3	0	5
v_4	2	2	3	4	4	4
v_5	6	2	0	0	1	6
$\ell(u_i)$	0	0	0	0	0	



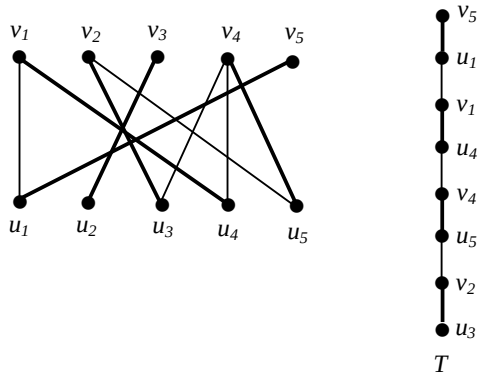
	u_1	u_2	u_3	u_4	u_5	$\ell(v_i)$
v_1	5	1	1	3	2	3
v_2	0	1	3	3	4	4
v_3	2	5	4	3	0	5
v_4	2	2	3	4	4	4
v_5	6	2	0	0	1	4
$\ell(u_i)$	2	0	0	0	0	



	u_1	u_2	u_3	u_4	u_5	$\ell(v_i)$
v_1	5	1	1	3	2	2
v_2	0	1	3	3	4	3
v_3	2	5	4	3	0	5
v_4	2	2	3	4	4	3
v_5	6	2	0	0	1	3
$\ell(u_i)$	3	0	0	1	1	

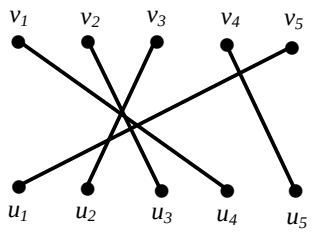


	u_1	u_2	u_3	u_4	u_5	$\ell(v_i)$
v_1	5	1	1	3	2	2
v_2	0	1	3	3	4	3
v_3	2	5	4	3	0	5
v_4	2	2	3	4	4	3
v_5	6	2	0	0	1	3
$\ell(u_i)$	3	0	0	1	1	



Gambar 4.5. *Matching* Maksimal Graf *G* dengan Algoritma Kuhn Munkres

Dari langkah di atas diperoleh *matching* sempurna yaitu $M = \{ v_5u_1, v_1u_4, v_4u_5, v_2u_3, v_3u_2\}$, dengan besar bobot maksimal sebesar 21. Jadi penempatan calon pekerja ke posisi jabatan pekerjaannya sebagai berikut :



Gambar 4.6. *Matching* Maksimal *M*

4.5. Kompleksitas Algoritma Kuhn Munkres

Keefisienan algoritma Kuhn Munkres untuk menyelesaikan input berukuran *nxn* dapat ditunjukkan dengan menggunakan kompleksitas. Langkah-langkah menentukan kompleksitas algoritma Kuhn Munkres sebagai berikut.

Misal dimasukkan sebuah *G* graf bipartit berbobot dengan *nxn vertex* ke dalam algoritma Kuhn Munkres. Setiap *vertex* dari graf *G* mempunyai *degree n*, maka langkah ke-1 punya kompleksitas sebesar $O(n^2)$. Pada langkah ke-2, pohon alternating disusun sebanyak *n* kali, dengan pengambilan *vertex* tunggal v_1 untuk digunakan sebagai akar dari pohon *alternating* yang akan disusun, mempunyai kompleksitas sebesar $O(n)$, sehingga kompleksitasnya menjadi $O(n^3)$. Setiap kali pohon *alternating* disusun pada langkah ke-3.c, $O(n^2)$ operasi telah dibentuk. Untuk setiap *vertex* tunggal *x* yang dipilih pada langkah ke-3.b, maka langkah ke-3.c dibentuk sebanyak $O(n)$ kali. Untuk menunjukkan hal ini, pada pengambilan *vertex* tunggal, setiap kali kembali ke langkah 3.c setelah yang pertama, untuk menyusun pohon *alternating* yang berakar di *x*, akan diperoleh lintasan *augmenting* atau kalau tidak dua *vertex* ditambahkan ke pohon *alternating* baru yang berakar di *x*. Pada saat pohon *alternating* yang berakar di *x* tidak memuat lintasan *augmenting* yang memiliki *2n-1 vertex*, dengan begitu kembali ke langkah 3.c paling banyak *n-1* kali. Oleh sebab itu, langkah 3.c keseluruhan

mempunyai kompleksitasnya sebesar $O(n^4)$. Langkah ke-4 dibentuk sebanyak $O(n^2)$ kali, setiap kali langkah ke-4 dijalankan, $O(n^2)$ langkah telah dibentuk. Pada saat langkah ke-5 di bentuk setiap kali maka langkah ke-4 telah selesai dijalankan, dan susunan dari G_ℓ punya kompleksitas $O(n^2)$, secara keseluruhan kompleksitas dari langkah ke-5 adalah $O(n^4)$. Oleh karena itu algorithma Kuhn-Munkres punya kompleksitas sebesar $O(n^4)$.

4.6. Aplikasi Algoritma Kuhn Munkres dalam Bahasa Pemgrograman

4.6.1. Langkah Penyusunan Program

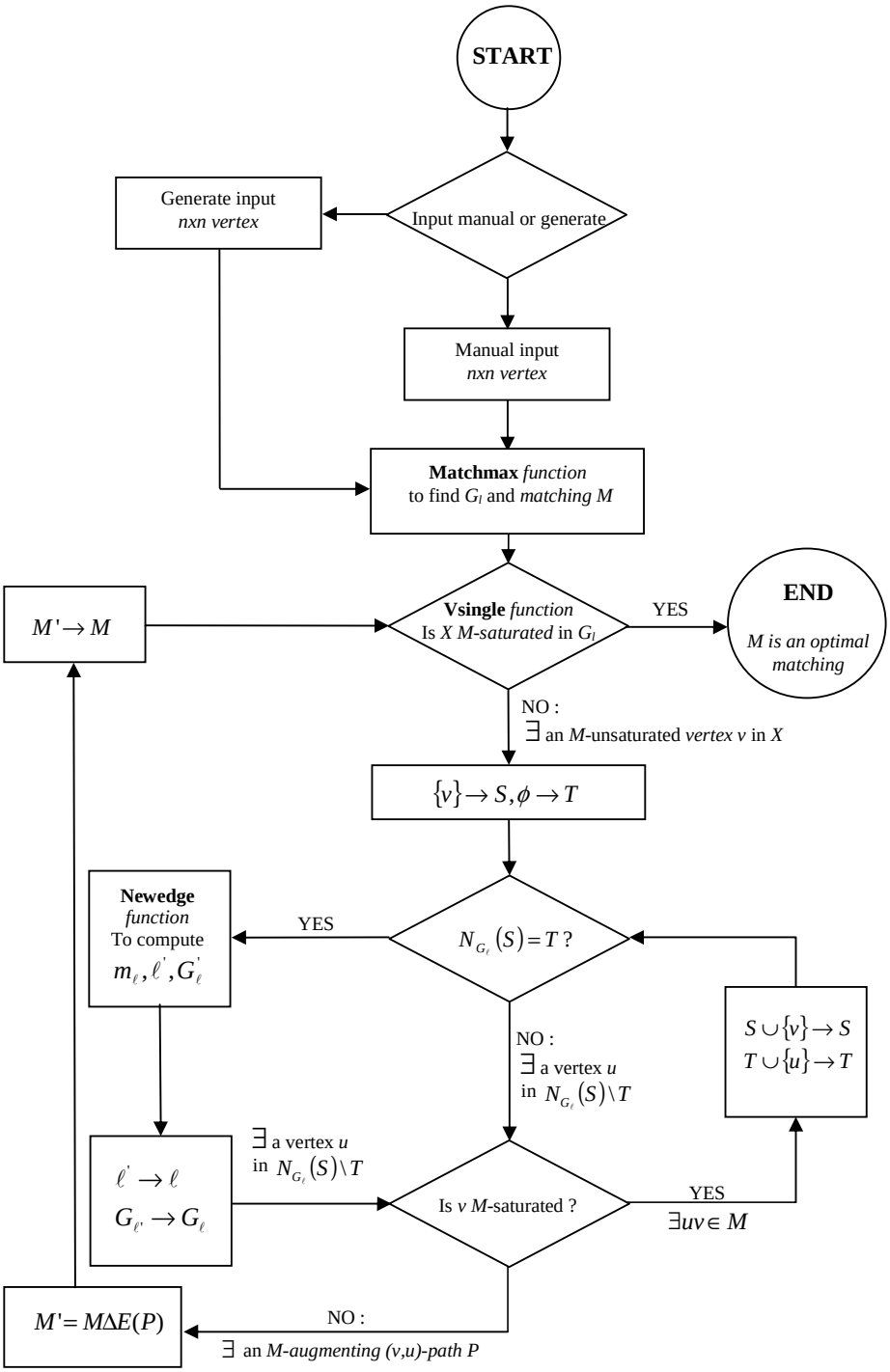
Penggunaan algoritma Kuhn Munkres dalam bahasa pemrograman digunakan untuk mempermudah dalam mencari *matching* maksimal berbobot, dengan jumlah *vertex* banyak.

Algoritma Kuhn Munkres dapat diterjemahkan ke dalam bahasa pemrograman dengan menggunakan software Matlab 6.1. Dalam tulisan ini, program diberi nama **KuhnMunkres**. Program ini memiliki diagram alir seperti tampak pada Gambar 4.7, diambil dari Bondy dan Murty (1976:89), dengan perubahan pada notasi dan proses *input* data.

Dalam program ini, terdapat satu program utama yaitu **KuhnMunkres** dan tiga *function* untuk mempermudah dalam pembuatan program yaitu **Matchmax**, **Vsingle** dan **Newedge**. Dalam program **KuhnMunkres** dimulai dengan memasukkan bobot *edge* yang menghubungkan dua partisi graf, dengan jumlah bobot *vertex nxn.* Proses *input* data diberikan dua pilihan yaitu memasukkan secara manual atau dengan *generate* data. Setelah memilih, untuk *input* data manual format input data diawali dengan tanda "[" dan diakhiri dengan "]", antara bobot *edge* satu dengan yang lain masih dalam satu baris diberi tanda ",", sedang untuk ganti baris diberi tanda ";". Setelah memasukkan bobot *vertex*, kemudian dicari bobot yang paling maksimal tiap barisnya, sehingga akan diperoleh subgraf perentang H_ℓ. Pada graf H_ℓ dapat dibentuk *matching* maksimal

dengan menggunakan *function* **Matchmax**. Pilih salah satu *matching* maksimal yaitu *Mlx*, kemudian dilakukan pengecekan apakah *Mlx* semua *vertex-nya* sudah masuk didalam *matching* tersebut, jika semua *vertex* sudah termasuk didalam *Mlx* maka *matching* tersebut adalah *matching* maksimal sehingga program akan berhenti dan *Mlx* adalah *matching* maksimal berbobot. Jika semua *vertex* tidak masuk dalam *Mlx*, maka terdapat *vertex* tunggal yang tidak masuk dalam *Mlx*, dengan menggunakan *function* **Vsinggle** akan diperoleh *vertex* tunggal yang belum masuk dalam *Mlx.*

Langkah berikutnya adalah penyusunan pohon *alternating*. Penyusunan pohon *alternating* dimulai dengan mendefinisikan variabel baru, yaitu *S* adalah *vertex* anggota V_1 yang masuk dalam penyusunan pohon *alternating*, *N(S)* adalah *vertex* anggota V_2 yang *adjecent* dengan anggota *S*, *T* adalah *vertex* anggota V_2 yang masuk dalam penyusunan pohon *alternating.* Langkah awal dalam penyusunan pohon *alternating* adalah dengan menggunakan *vertex* tunggal misal v_1 sebagai akar pohon *alternating*. Dengan definisi awal $S \leftarrow \{v_1\}, T \leftarrow \phi$. Seperti yang tertera pada diagram alir pada Gambar 4.7, dilakukan pengecekan apakah $N_{G_\ell}(S) = T$, jika $N_{G_\ell}(S) = T$, maka dilakukan pelabelan baru dengan menghitung $m_\ell, \ell', G_{\ell'}$. Pelabelan baru dilakukan dengan menggunakan *function* **Newedge**, dengan output pelabelan baru dan graf $G_{\ell'}$ yang baru. Pada graf $G_{\ell'}$ terdapat *vertex u* dalam $N_{G_{\ell'}}(S) \backslash T$. Jika $N_{G_\ell}(S) \neq T$, maka terdapat *vertex u* dalam $N_{G_\ell}(S) \backslash T$. Kemudian dicek apakah *vertex u* adalah *vertex matching* dalam *Mlx*, jika *vertex u* adalah *vertex matching* maka terdapat *vertex v*, dimana *uv* adalah *edge matching* dalam *Mlx*. Dengan penambahan *vertex u* dan *vertex v* dalam pohon *alternating* maka anggota himpunan *S* menjadi $S \cup \{v\} \rightarrow S$, dan anggota himpunan *T* menjadi $T \cup \{u\} \rightarrow T$. Dengan penambahan anggota pada himpunan *S* maka kembali pada langkah pengecekan apakah $N_{G_\ell}(S) = T$.



Gambar 4.7. Diagram Alir Program **KuhnMunkres**

Jika pada saat pengecekan *vertex u* bukan merupakan anggota *vertex matching* maka terdapat lintasan *augmenting* dari akar pohon *alternating* v_1 sampai *vertex u.* Sehingga dapat dibentuk *matching* baru dengan $Mla = (Mlx - E(P)) \cup M(Ep)$. Setelah ditemukan *matching* baru, kemudian kembali pada langkah pengecekan apakah semua *vertex* sudah termasuk dalam *matching* baru tersebut, jika semua sudah termasuk maka *Mla* adalah *matching* maksimal berbobot. Jika masih terdapat *vertex* tunggal maka ulangi langkah dari pengambilan *vertex* tunggal sebagai akar pohon *alternating* sampai semua *vertex* termasuk dalam *matching* terbaru.

Penghitungan bobot maksimal dilakukan dengan menjumlahkan bobot *edge* pada *matching* maksimal yang telah diperoleh. Listing program dapat dilihat pada halaman lampiran.

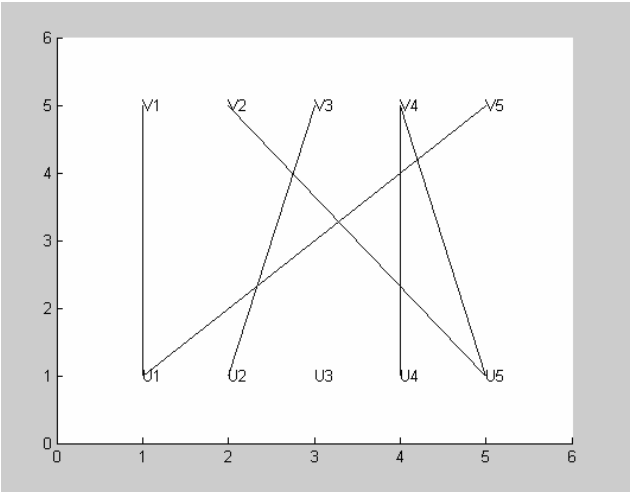
4.6.2. Analisis Hasil Pemrograman

Uraian pada subbab di atas akan lebih jelas apabila diterapkan dalam suatu contoh graf. Misal graf *G* adalah graf pada Gambar 4.5.a. Dengan input bobot *vertex n x n,* yaitu : G = [5, 1, 1, 3, 2; 0, 1, 3, 3, 4; 2, 5, 4, 3, 0; 2, 2, 3, 4, 4; 6, 2, 0, 0, 1], dengan hasil output sebagai berikut :

```
==== Matrik Bobot G ================
G =
    5    1    1    3    2
    0    1    3    3    4
    2    5    4    3    0
    2    2    3    4    4
    6    2    0    0    1
```

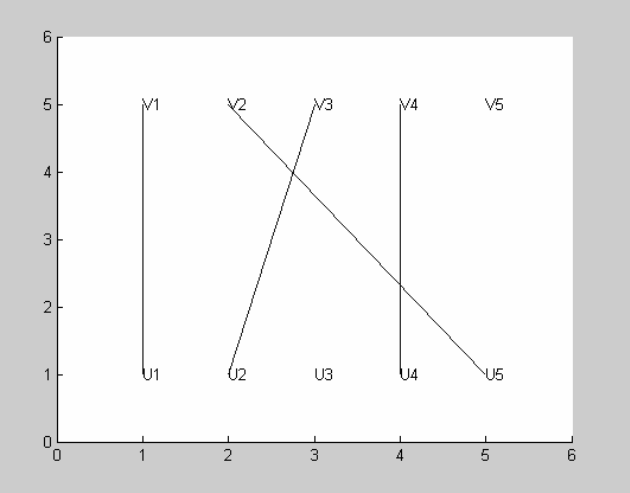
```
==== Graf Dasar ====================
    1    1
    2    5
    3    2
    4    4
    4    5
    5    1
==== Graf Matching Awal terpilih ===
    1    1
    2    5
    3    2
    4    4
==== Graf Gl akhir =================
    1    1
    2    5
    3    2
    4    4
    4    5
    5    1
    1    4
    2    3
    4    3
==== Graf Match Maks akhir ==========
    3    2
    5    1
    1    4
    4    5
    2    3
==== Bobot Matching Maksimal =====
   21
```

Apakah ingin menampilkan Graf Hl (Y/N) ? 'y'



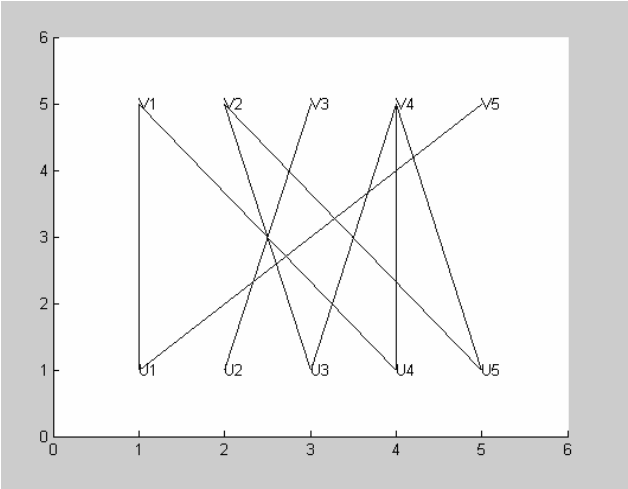
Gambar 4.8. Graf H_ℓ

Apakah ingin menampilkan Matching Maks Awal (Y/N) ? ‘y’



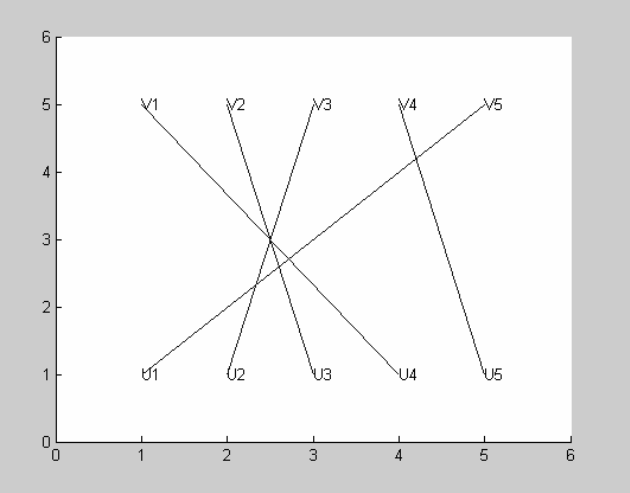
Gambar 4.9. Graf *Matching* Awal

Apakah ingin menampilkan Graf Gl (Y/N) ? 'y'



Gambar 4.10. Graf G_ℓ '

Apakah ingin menampilkan Final Graf Matching Maks (Y/N) ? 'y'



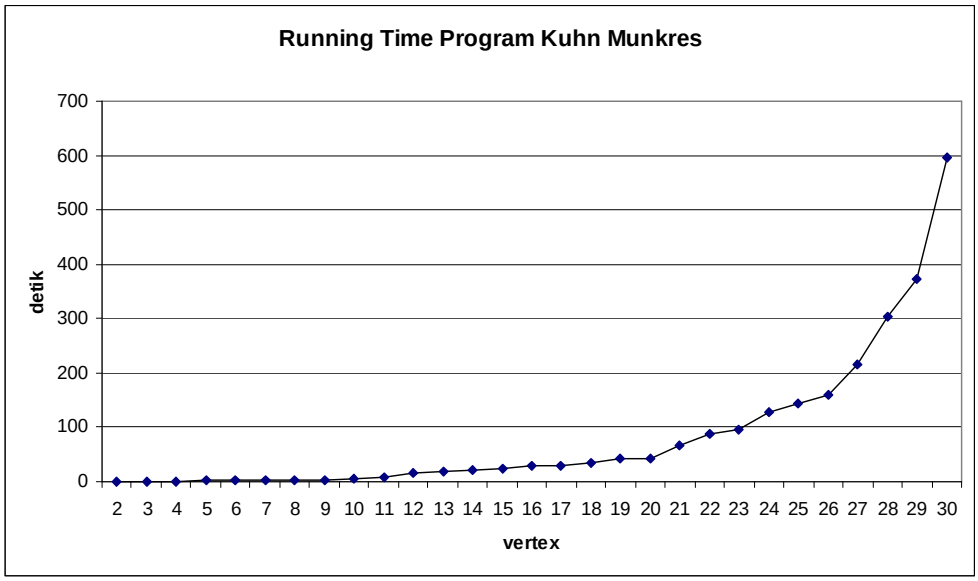
Gambar 4.11. Graf *Matching* Maksimal

Hasil penghitungan secara manual dan hasil menggunakan program nilainya sama, bobot *matching* maksimal bipartit adalah 21.

4.6.3. Kompleksitas *Running Time* Program

Pengecekan *running time* program **KuhnMunkres** dilakukan dengan mencatat waktu yang diperlukan untuk menyelesaikan suatu *input* data. *Input vertex nxn* dimulai dari $n = 2$ sampai dengan $n = 30$. Spesifikasi komputer yang digunakan adalah komputer Pentium III 733 MHz, dengan memori 128 MB, dan dengan *operating system* Windows 98. Hasil *running time* dapat dilihat pada Gambar 4.12.

Grafik *running time* pada Gambar 4.12 belum bisa menunjukkan bahwa *running time* program **KuhnMunkres** sesuai dengan penghitungan *running time* algoritma Kuhn Munkres yang diukur dari jumlah tahapan komputasi yang dibutuhkan untuk menjalankan algoritma tersebut . Komputer dengan arsitektur yang berbeda akan berbeda pula lama waktu yang diperlukan untuk menyelesaikan suatu input data.



Gambar 4.12. Grafik *Running Time* Program **KuhnMunkres**

BAB V
KESIMPULAN DAN SARAN

5.1. Kesimpulan

Berdasar uraian pada bab pembahasan, maka dapat disimpulkan beberapa hal sebagai berikut **:**

1. *Matching* maksimal pada graf bipartit berbobot dapat diperoleh dengan menggunakan Algoritma Kuhn Munkres.
2. Algoritma Kuhn Munkres memiliki kompleksitas sebesar $O(n^4)$.
3. Algoritma Kuhn Munkres dapat digunakan dalam pembuatan program dengan bantuan *software* Matlab 6.1 untuk mempermudah dalam mencari matching maksimal pada graf bipartit berbobot.

5.2. Saran

1. *Matching* maksimal yang dicari dalam penulisan ini adalah *matching* maksimal pada graf bipartit berbobot, disarankan pencarian *matching* maksimal pada sembarang graf.
2. Selain menggunakan *software* Matlab 6.1, terdapat *software* pemrograman yang lain, seperti Delphi yang bisa menampilkan output yang lebih optimal dalam visualnya. Oleh karena itu, disarankan untuk menggunakan *software* tersebut untuk mendapatkan tampilan visual yang lebih menarik.

DAFTAR PUSTAKA

Bondy, J.A. and U. S. Murty., *Graph Theory with Applications*, American Elsevier, New York, 1976.

Chartrand, G. and Lesniak, L., *Graph and Digraph*, Wadsworth & Brooks / Cole Advanced Book & Software, Pacific Grove, California, 1986.

Chartrand, G. and Oellerman, O. R., *Applied and Algorithmic Graph Theory*, McGraw-Hill Inc., New York, 1993.

Fletcher, P., H. Hoyle and C. W. Patty, *Foundations of Discrete Mathematics*, PWS-Kent Publishing Company, Boston, 1991.

Johnsonbough, R., *Discrete Mathematics*, revised ed., Macmillan Publishing Company, New York, 1991.

Rosen, K. H., *Discrete Mathematics and Its Applications*, fifth ed., McGraw-Hill Inc., New York, 2003.

Toroslu, I. H. and Ucoluk G., *Incremental Assignment Problem*, Preprint submitted to Elsevier Science, Department of Computer Engineering Middle East Technical University, Ankara, Turkey, May 2006.

LAMPIRAN

1. Listing Program **KuhnMunkres.m**

```
clear all;
upilih=1;
while upilih==1
    disp('   ');
    disp('Pilih cara input G secara manual atau Generate G otomatis');
    disp('==========================================');
    disp('      0. Input G secara Manual');
    disp('      1. Generate G otomatis');
    disp('-----------------------------------------------------------------');
    pil=input('Pilih 0 atau 1 : ');

    if pil==0
        n=input('\nMasukan nilai n (ukuran matriks G : nxn) = ');
        disp('Cara input G :   ');
        disp('- awali dengan tanda [');
        disp('- Pisahkan tiap elemen dengan SPASI atau tanda , (koma)');
        disp('- untuk ganti baris, gunakan tanda ;');
        disp('- akhiri dengan tanda ]');
        disp('   ');
        clear gs;
        gs=input('');
        G=gs;

        if size(G,1)==n & size(G,2)==n
            upilih=0;
        else
            disp('MAAF input G salah !!');
            upilih=1;
```

```
        end
    elseif pil==1
        n=input('\nMasukan nilai n (ukuran matriks G : nxn) = ');
        G=randint(n,n,[1 5]);
        upilih=0;
    else
        disp('MAAF Pilihan Anda Salah. Pilih 0 atau 1 !');
        upilih=1;
    end
end

FbMfinal=0;
bMfinal=0;
cekcek=1;
icek=0;
while cekcek==1

all_u=[1:n];
all_v=[1:n];
MaxU(1:n)=0;
%%== Mencari Maksimal tiap baris di G sbg Spanning subgraf
i=1;
Hl=[];
while i<=n
    m=max(G(i,:));
    MaxV(i)=m;
    clear iu;
    iu=find(G(i,:)==m);
    j=1;
    while j<=length(iu)
        Hl=[Hl;[i,iu(j)]];
```

```
        j=j+1;
    end
    i=i+1;
end
%%== mendefinisikan Gl sebagai graf dasar dari Hl
Gl=[];Ml=[];M=[];
Gl=Hl;
[Ml,M,nma]=matchmax(Gl);
%% Dari semua matching maksimal awal yg fix,
%% dipilih salah satu Matching maks secara random
clear Mla;
r=randint(1,1,[1 nma]);
%r
Mlx=Ml{r}; %% Matching maks terpilih
%==========### MULAI TAHAP PENGUJIAN###============
%STEP 1. Cek apakah semua vertex dlm Matching maks terpilih (Mla)
Mla=Mlx;
ulang1=1;
coba=1;
rv=[];
while ulang1==1
    %%----------A-----------------
    vpake=unique(Mla(:,1));
    upake=unique(Mla(:,2));
    v_cek=isempty(setdiff(all_v,vpake));
    u_cek=isempty(setdiff(all_u,upake));

    if (v_cek==1 & u_cek==1 ) %% Matching maksimal
        if (all(ismember(Mla,Gl,'rows'))==1)
            ulang1=0;
            break;
```

```
    end
else
    SE=[];
    SV=[];
    [SE SV]=vsingle(Gl,Mla); %% Single Edge dan single vertex
    % Memilih single vertex
    T=[];S=[];NS=[];
    if size(SE,1)==1
        Ss=SE(1,SV);
    else
        if rv~=[]
            rva=[];
            rva=[1:size(SE,1)];
            rvs=setdiff(rva,rv);
            if rvs~=[]
                rv=rvs(1);
            else
                rv=randint(1,1,[1 size(SE,1)]);
            end

        else
            rv=randint(1,1,[1 size(SE,1)]);
        end
        Ss=SE(rv,SV);
    end
    S=Ss;
    ulang2=1;
    coba2=1;
    while ulang2==1
        %cek apakah NS=T?
        iNS=[];NSs=[];NSnew=[];
```

```
iNS=find(Gl(:,1)==S(length(S)));
NSs=Gl(iNS,2);
NSnew=setdiff(NSs',NS);
NS=[NS,NSnew];
if all(ismember(NS,T))
   %hitung ml
   Vnew=[];
   MVnew=[];MUnew=[];
   [Vnew MVnew MUnew]=newedge(G,S,T,MaxV,MaxU);
   MaxV=MVnew;
   MaxU=MUnew;

   for j=1:1:size(Vnew,1)
      if ~ismember(Vnew(j,:),Gl,'rows')
         Gl=[Gl;Vnew(j,:)];
      end
   end
   % find u di NS tp tdk di T
   break;
else
   % find u di NS tp tdk di T
end
% find u di NS tp tdk di T
NST=[];Ts=[];
NST=setdiff(NS,T);
Ts=setdiff(NST,T);
if Ts~=[]
   iTs=randint(1,1,[1 length(Ts)]);
   %for iTs=1:1:length(Ts)
      %G(S(length(S)),Ts(iTs))
      if G(S(length(S)),Ts(iTs))~=0
```

```
T=[T,Ts(iTs)];
%cek apakah Ts anggota Mla
if ismember(Ts(iTs),Mla(:,2))
   % ada v elemen S
   Ss=[];
   Snew=[];
   iS=find(Mla(:,2)==Ts(iTs));
   Ss=Mla(iS,1);
   Snew=[];
   Snew=setdiff(Ss,S);
   if Snew~=[]
      if length(Snew)>1
         S=[S,Snew(randint(1,1,[1 length(Snew)]))];
      else
         S=[S,Snew];
      end
   end
   %% kembali ke : cek apakah NS=T?
else
   % path dibalik karena lintasan augmenting ditemukan
   if (length(S)+length(T))>=2
      %Mla=PA gabung (Mla-PA)
      PA=[];PA1=[];PA2=[];

      if mod(length(S)+length(T),2)==0
         for iPA1=1:1:length(S)
            PA1=[PA1;[S(iPA1) T(iPA1)]];
         end
      else
         for iPA1=1:1:length(T)-1
            PA1=[PA1;[S(iPA1+1) T(iPA1)]];
```

```
            end
        end
        PA=PA1;

        if PA~[]
            MPAs=[];
            MPAs=setdiff(Mla,PA,'rows');
          %iMPA=find(MPAs(:,1)==setdiff(MPAs(:,1),PA(:,1)));
            MPA=[];
            %MPA=MPAs(iMPA,:);
            for k=1:1:size(MPAs,1)
                if ~ismember(MPAs(k,1),PA(:,1)) &
                  ~ismember(MPAs(k,2),PA(:,2))
                  MPA=[MPA;MPAs(k,:)];
                end
            end
            Mla=[];
            Mla=[MPA;PA];
        end
    end

    ulang2=0;
    break;

  end

  else
    break;
  end

end
```

```
            coba2=coba2+1;
            if (coba2>=10)
                ulang2=0;
            end
        end  % end while ulang2

    end  % End else v_cek==1 & u_cek==1

    coba=coba+1;
    if (coba>=30)
        ulang1=0;
    end
end  %End while ulang1

if ((all(ismember(Mla,Gl,'rows')))==1)
    Mfinal=[];
    Mfinal=Mla;
    bMfinal=0;
    for i=1:1:size(Mfinal,1)
        bMfinal=bMfinal+G(Mfinal(i,1),Mfinal(i,2));
    end

    if bMfinal>FbMfinal
        if icek>1
            FS=S;
            FT=T;
            FPA=PA;
        end
        FHl=Hl;
        FGl=Gl;
        FMlx=Mlx;
```

```
        FMla=Mla;
        FbMfinal=bMfinal;
    end
end
icek=icek+1;
if (FbMfinal>0)
    cekcek=0;
    break;
end
end

%% show output
disp('   ');
disp('==== Matrik Bobot G ================');
G
disp('   ');
disp('==== Graf Dasar ====================');
disp(FHl);
%sprintf('V%d U%d\n',Hl(:,1),Hl(:,2))
disp('==== Graf Matching Awal terpilih ===');
disp(FMlx);
%sprintf('V%d U%d\n',Mlx(:,1),Mlx(:,2))
disp('==== Graf Gl akhir =================');
disp(FGl);
%sprintf('V%d U%d\n',Gl(:,1),Gl(:,2))
disp('==== Graf Match Maks akhir ==========');
disp(FMla);
%sprintf('V%d U%d\n',Mfinal(:,1),Mfinal(:,2))
disp('==== Bobot Matching Maksimal =====');
disp(FbMfinal);
```

```
%% show graf picture
fig=1;
while fig==1
  figHl=input('\nApakah ingin menampilkan Graf Hl (Y/N) ? ' );
  if (upper(figHl)=='Y') | (upper(figHl)=='N')
     if upper(figHl)=='Y'
        fig=0;
        gmatching(FHl,n);
     else
        fig=0;
     end
  else
      fig=1;
      disp('\nMAAF INPUT SALAH !!\n');
  end
end

fig=1;
while fig==1
   figMlx=input('\nApakah ingin menampilkan Matching Maks Awal
(Y/N) ? ' );
   if (upper(figMlx)=='Y') | (upper(figMlx)=='N')
      if upper(figMlx)=='Y'
         fig=0;
         gmatching(FMlx,n);
      else
         fig=0;
      end
   else
      fig=1;
      disp('\nMAAF INPUT SALAH !!\n');
```

```
    end
end

fig=1;
while fig==1
    figGl=input('\nApakah ingin menampilkan Graf Gl (Y/N) ? ' );
    if (upper(figGl)=='Y') | (upper(figGl)=='N')
        if upper(figGl)=='Y'
            fig=0;
            gmatching(FGl,n);
        else
            fig=0;
        end
    else
        fig=1;
        disp('\nMAAF INPUT SALAH !!\n');
    end
end

fig=1;
while fig==1
    figMF=input('\nApakah ingin menampilkan Final Graf Matching Maks
(Y/N) ? ' );
    if (upper(figMF)=='Y') | (upper(figMF)=='N')
        if upper(figMF)=='Y'
            fig=0;
            gmatching(FMla,n);
        else
            fig=0;
        end
    else
```

```
        fig=1;
        disp('\nMAAF INPUT SALAH !!\n');
    end
end
```

2. Listing Program Function **Matchmax**

```
function [Mmax,Mall,nfix]=matchmax(Gl);
%=======### Function utk mencari Matching Maksimal ##======
% ------------------------------- Parameter Input ------------------------------
% Gl : Spanning subgraf Gl
% ------------------------ Return atau Output function ------------------------
% Mmax: Hanya Graf Matching Maksimal dari Gl
% Mall: Semua kemungkinan Graf Matching dari Gl
% nfix: banyaknya graf matching maksimal (banyaknya Mmax)
%==================================================
%-------------------- LANGKAH ALGORITMA --------------------------

ulangM=1;
im=1;
starg=1;
%% 1. Mencari semua kemungkinan Matching Maksimal ==> M
while ulangM==1
%% 1.1. dari atas ke bawah / indeks kecil ke besar
    ig=starg;
    Mx=[];
    for i=1:1:size(Gl,1)
        clear fv;
        clear fu;
        if Mx==[] Mx=[Mx;Gl(ig,:)];
        else
            fv=find(Mx(:,1)==Gl(ig,1));
```

```
            fu=find(Mx(:,2)==Gl(ig,2));
            if fv==[] & fu==[]
                Mx=[Mx;Gl(ig,:)];
            end
        end
        ig=ig+1;
        if ig>size(Gl,1)
            ig=ig-size(Gl,1);
        end
    end
    M{im}=Mx;
    im=im+1;
%% 1.2. dari bawah ke atas / indeks besar ke kecil
    ig=starg;
    Mx=[];
    for j=size(Gl,1):-1:1
        clear fv;
        clear fu;
        if Mx==[] Mx=[Mx;Gl(ig,:)];
        else
            fv=find(Mx(:,1)==Gl(ig,1));
            fu=find(Mx(:,2)==Gl(ig,2));
            if fv==[] & fu==[]
                Mx=[Mx;Gl(ig,:)];
            end
        end
        ig=ig-1;
        if ig<1
            ig=ig+size(Gl,1);
        end
    end
```

```
    M{im}=Mx;
    im=im+1;

    starg=starg+1;
    if starg>size(Gl,1)
        ulangM=0;
    end
end
%% 2. Menghitung banyaknya elemen setiap M
for i=1:1:size(M,2)
    eM(i)=size(M{i},1);
end
%% 3. Matching Maksimal awal dari M (anggota/elemennya paling
banyak)
clear fmax;
fmax=find(eM==max(eM));
for i=1:1:length(fmax)
    Ma{i}=M{fmax(i)};
end
%% 4. Membuang matching maksimal awal yg anggotanya identik,
sehingga menjadi Matching Maksimal awal yg Fix
ulangfm=1;
fm=fmax;
i=1;
j=1;
while ulangfm==1
    ima(j)=fm(j);
    ulangima=1;
    i=j;
    while ulangima==1
        if (i+1<=length(fm))
```

```
            if size((intersect(M{fm(j)},M{fm(i+1)},'rows')),1)==max(eM)
                fm(i+1)=[];
                i=j;
            else
                i=i+1;
            end
        else ulangima=0;
        end
    end
    if j<length(fm)
        j=j+1;
        i=j;
    else
        ulangfm=0;
    end
end
%% 5. Matching Maksimal awal Fix ==> Ml
clear Ml;
for i=1:1:length(ima)
    Ml{i}=M{ima(i)};
end;
%%============== Return atau hasil function ==============
Mmax=Ml;
Mall=M;
nfix=length(ima);
%%==============================================
```

3. Listing Program Function **Vsingle**

```
function [E,V]=vsingle(Gl,Ml);
%====### Function utk mencari Singgle Vertex dan Edge nya ##===
%% --------------------------- Parameter Input -------------------------------
%% Gl : Spanning subgraf Gl
%% Ml : Matching Maksimal dari Gl yang terpilih
%% ------------------------------------------------------------------------------
%% ------------------------ Return atau Output function ----------------------
%% E : Edge dari single vertex dalam Gl
%% V : Single Vertex (vertex dalam Gl yang tidak terpakai dalam Ml
%%==============================================
isv=find(ismember(Gl(:,1),Ml(:,1))==0);
if isv~=[]
   SE=Gl(isv,:);%% Edge dari Single vertex
   sv=1; %% single vertex pada v
else
   isv=find(ismember(Gl(:,2),Ml(:,2))==0);
   SE=Gl(isv,:);%% Edge Single vertex
   sv=2; %% single vertex pada u
end
%% Return / Output
E=SE;
V=sv;
```

4. Listing Program Function **Newedge**

```
function [Vnew,MVnew,MUnew]=newedge(G,Tv,Tu,MaxV,MaxU);
%= Function untuk mencari edge baru dalam Gl (Matching maksimal) ==
% ------------------------------- Parameter Input ---------------------------------
% G : Matriks G
% Tv : vertek v yang dipakai dalam pohon alternating T
% Tu : vertek u yang dipakai dalam pohon alternating T
% MaxV : Nilai maksimal graf G utk setiap baris V (label V)
% MaxU : Nilai maksimal graf G utk setiap kolom U (label U)
% ------------------------- Return atau Output function -------------------------
% Vnew : pasangan vertek (v,u) atau edge baru
% MVnew: Nilai maksimal graf G utk setiap baris V (label V) yang baru
% MUnew: Nilai maksimal graf G utk setiap kolom U (label U) yang baru
%==================================================

all_v=[1:length(MaxV)];
all_u=[1:length(MaxU)];
if size(all_v,1)<size(all_v,2)
   all_v=all_v';
end
if size(all_u,1)<size(all_u,2)
   all_u=all_u';
end
VT=intersect(Tv,all_v);
UT=setdiff(all_u,Tu);
vml_all=[];
for i=1:1:length(VT)
   for j=1:1:length(UT)
      vml=MaxV(VT(i))+MaxU(UT(j))-G(VT(i),UT(j));
      vml_all=[vml_all;vml];
      val_ml(i,j)=vml;
```

```
    end
end
min_ml=min(vml_all);
if min_ml<=0
    min_ml=min(setdiff(vml_all,0));
end
if min_ml~=[]
    [ix,iy]=find(val_ml==min_ml);
    Vnew=[];
    for i=1:1:length(ix)
        Vnew=[Vnew;[VT(ix(i)),UT(iy(i))]];
    end
    VnT=VT;
    MVnew=MaxV;
    if size(VnT,1)>1
        for i=1:1:length(VnT)
            MVnew(VnT(i))=MaxV(VnT(i))-min_ml;
        end
    else
        MVnew(VnT)=MaxV(VnT)-min_ml;
    end
    UnT=intersect(Tu,all_u);
    MUnew=MaxU;
    for i=1:1:length(UnT)
        MUnew(UnT(i))=MaxU(UnT(i))+min_ml;
    end
else
    Vnew=[];
    MVnew=MaxV;
    MUnew=MaxU;
end
```