Minimum Spanning Tree dan Desain Jaringan

Minimum Spanning Tree (MST) merupakan salah satu konsep dalam Teori Graf yang mempunyai peranan besar dalam masalah optimisasi, terutama masalah desain dan optimisasi jaringan. Dalam optimisasi jaringan, sebelum suatu masalah diselesaikan, masalah tersebut direpresentasikan atau divisualisasikan dahulu agar dapat diketahui struktur masalah yang sebenarnya. Hal ini disebut dengan Desain Jaringan (Network Design). Dalam Network Design, konsep yang paling sering digunakan untuk mengilustrasikan atau merepresentasikan masalah adalah adalah konsep-konsep dari Teori Graf. Teori Graf digunakan karena fleksibilitas untuk merepresentasikan masalah sehingga hampir setiap masalah dalam optimisasi jaringan dapat direpresentasikan dengan konsep teori graf secara akurat.

Buku ini didesain sebagai bahan acuan alternatif tambahan bagi mahasiswa Matematika, Ilmu Komputer atau Teknik Informatika yang akan mempelajari konsep-kosep dasar yang berhubungan dengan mata kuliah Optimisasi, Riset Operasi, Matematika Diskrit, Graf dan Desain Jaringan. Selain itu, buku ini berisi diskusi tentang salah satu masalah yang muncul dalam desain jaringan yaitu masalah Degree Constrained Minimum Spanning Tree (DCMST), yang menggunakan Minimum Spanning Tree sebagai backbone masalah. Beberapa penelitian yang telah menginvestigasi metode-metode untuk menyelesaikan masalah DCMST akan didiskusikan, baik metode exact maupun heuristic. Sehingga, buku ini juga dapat dijadikan sebagai bahan rujukan bagi mereka yang ingin melakukan penelitian yang lebih mendalam tentang masalah DCMST.

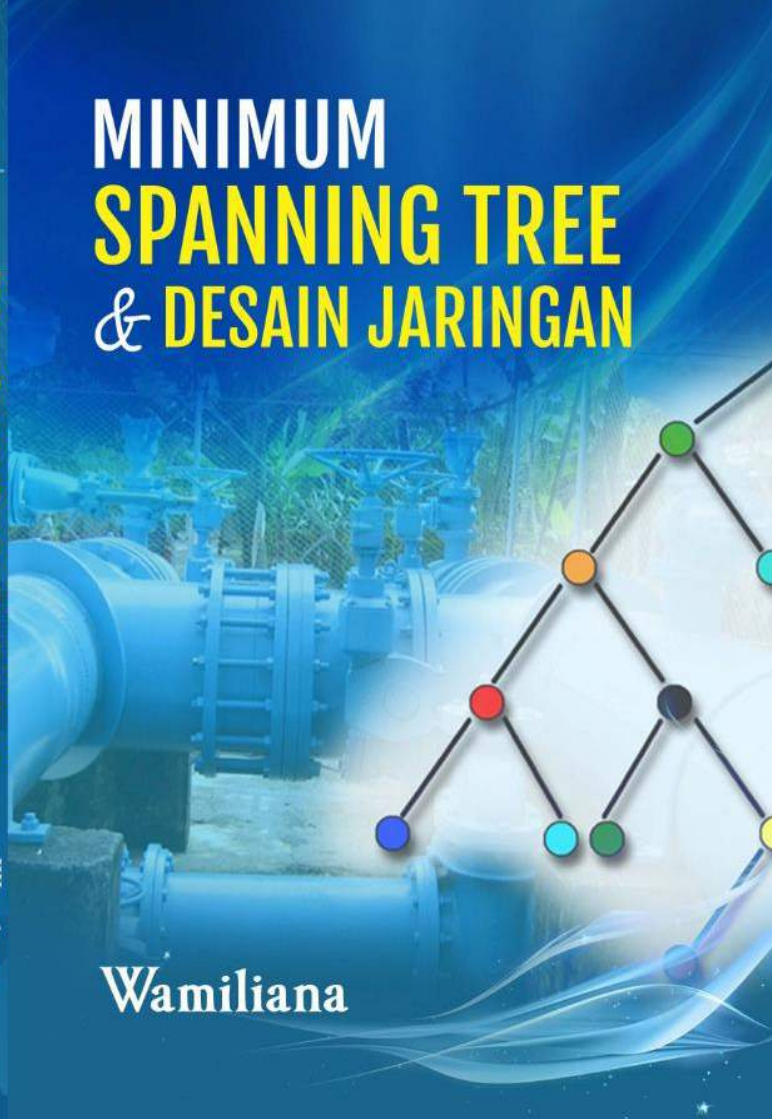
PUSAKA MEDIA
penerbit pusaka
pusakamedia@gmail.com
@pusaka_media

ISBN 978-623-418-030-5
9 786234 180305

Minimum Spanning Tree dan Desain Jaringan

Wamiliana

MINIMUM SPANNING TREE & DESAIN JARINGAN



Wamiliana

MINIMUM SPANNING TREE & DESAIN JARINGAN

Undang-undang Republik Indonesia Nomor 28 tahun 2014 tentang Hak Cipta
Lingkup Hak Cipta
Pasal 1
Hak Cipta adalah hak eksklusif pencipta yang timbul secara otomatis berdasarkan prinsip deklaratif setelah suatu ciptaan diwujudkan dalam bentuk nyata tanpa mengurangi pembatasan sesuai dengan ketentuan peraturan perundang-undangan.
Ketentuan Pidana Pasal 113
(1) Setiap Orang yang dengan tanpa hak melakukan pelanggaran hak ekonomi sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf i untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 1 (satu) tahun dan/atau pidana denda paling banyak Rp 100.000.000 (seratus juta rupiah).
(2) Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf c, huruf d, huruf f, dan/atau huruf h untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 3 (tiga) tahun dan/atau pidana denda paling banyak Rp 500.000.000,00 (lima ratus juta rupiah).
(3) Setiap Orang yang dengan tanpa hak dan/atau tanpa izin Pencipta atau pemegang Hak Cipta melakukan pelanggaran hak ekonomi Pencipta sebagaimana dimaksud dalam Pasal 9 ayat (1) huruf a, huruf b, huruf e, dan/atau huruf g untuk Penggunaan Secara Komersial dipidana dengan pidana penjara paling lama 4 (empat) tahun dan/atau pidana denda paling banyak Rp 1.000.000.000,00 (satu miliar rupiah).
(4) Setiap Orang yang memenuhi unsur sebagaimana dimaksud pada ayat (3) yang dilakukan dalam bentuk pembajakan, dipidana dengan pidana penjara paling lama 10 (sepuluh) tahun dan/atau pidana denda paling banyak Rp 4.000.000.000,00 (empat miliar rupiah).

MINIMUM SPANNING TREE & DESAIN JARINGAN

Wamiliana

PUSAKA MEDIA

Perpustakaan Nasional RI:
Katalog Dalam Terbitan (KDT)

Minimum Spanning Tree dan Desain Jaringan

Penulis:
Wamiliana

Desain Cover
Shela Malinda

Layout
Pusaka Media Design

x + 127 hal : 14.5 x 21 cm
Cetakan, Januari 2022

ISBN: 978-623-418-030-5

Penerbit
PUSAKA MEDIA
Anggota IKAPI
No. 008/LPU/2020

Alamat
Jl. Endro Suratmin, Pandawa Raya. No. 100
Korpri Jaya Sukarame Bandarlampung
082282148711
email : cspusakamedia@yahoo.com
Website : www.pusakamedia.com

KATA PENGANTAR

Minimum Spanning Tree (MST) merupakan salah satu konsep dalam Teori Graf yang mempunyai peranan besar dalam masalah optimisasi, terutama masalah desain dan optimisasi jaringan. Dalam optimisasi jaringan, sebelum suatu masalah diselesaikan, masalah tersebut direpresentasikan atau divisualisasikan dahulu agar dapat diketahui struktur masalah yang sebenarnya. Hal ini disebut dengan Desain Jaringan (*Network Design*). Dalam *Network Design*, konsep yang paling sering digunakan untuk mengilustrasikan atau merepresentasikan masalah adalah konsep-konsep dari Teori Graf. Teori Graf digunakan karena fleksibilitas untuk merepresentasikan masalah sehingga hampir setiap masalah dalam optimisasi jaringan dapat direpresentasikan dengan konsep teori graf secara akurat.

Buku ini didesain sebagai bahan acuan alternatif tambahan bagi mahasiswa Matematika, Ilmu Komputer atau Teknik Informatika yang akan mempelajari konsep-kosep dasar yang berhubungan dengan mata kuliah Optimisasi, Riset Operasi, Matematika Diskrit, Graf dan Desain Jaringan. Selain itu, buku ini berisi diskusi tentang salah satu masalah yang muncul dalam desain jaringan yaitu masalah *Degree Constrained Minimum Spanning Tree* (DCMST), yang

menggunakan *Minimum Spanning Tree* sebagai *backbone* masalah. Beberapa penelitian yang telah menginvestigasi metode-metode untuk menyelesaikan masalah DCMST akan didiskusikan, baik metode *exact* maupun *heuristic*. Sehingga, buku ini juga dapat dijadikan sebagai bahan rujukan bagi mereka yang ingin melakukan penelitian yang lebih mendalam tentang masalah DCMST.

Untuk mencapai tujuan tersebut, buku ini didesain terbagi atas 4 bab dengan rincian sebagai berikut:

Bab I: Konsep Dasar Teori Graf. Pada bab ini akan dijelaskan berbagai konsep dasar dari Teori Graf yang banyak digunakan dalam proses desain jaringan. Konsep-konsep dasar tersebut dijabarkan dengan cukup jelas dengan contoh-contoh agar mahasiswa dapat memahami apa yang didiskusikan dalam bab tersebut.

Bab II: *Minimum Spanning Tree*. Pada bab ini akan didiskusikan beberapa algoritma untuk menentukan suatu *Minimum Spanning Tree*. Terdapat banyak algoritma untuk menyelesaikan *Minimum Spanning Tree*, akan tetapi pada buku ini hanya akan didiskusikan tiga algoritma untuk menyelesaikan *Minimum Spanning Tree* yaitu Algoritma Sollin atau Boruvka yang merupakan algoritma yang pertama kali dibuat untuk menyelesaikan *Minimum Spanning Tree*. Selain Algoritma Sollin, dua algoritma lain yang sangat umum digunakan yaitu Algoritma Kruskal dan Prim akan dijelaskan dengan teknik yang berbeda dari yang selama ini ada di literature. Pendekatan tersebut adalah dengan menggunakan table dimana dalam

setiap iterasi apa saja yang terjadi dapat dilihat pada tabel.

Bab III: Optimisasi Kombinatorial. Pada bab ini akan didiskusikan tentang optimisasi kombinatorial, metode-metode untuk menyelesaikan masalah optimisasi kombinatorial yaitu metode *exact* (antara lain metode *branch and bound*, *cutting plane*, dan relaksasi Lagrange), metode *heuristic* (yang dalam buku ini akan didiskusikan tentang *Tabu Search*), dan desain jaringan.

Bab IV: *Degree Constrained Minimum Spanning Tree* (DCMST). Penerapan dari *Minimum Spanning Tree* pada desain dan optimisasi jaringan, antara lain pada masalah *Degree Constrained Minimum Spanning Tree* (MST) yang mengunakan *Minimum Spanning Tree* sebagai *backbone* nya. DCMST ini merupakan masalah reliabilitas (keterhandalan) dari suatu jaringan yang berhubungan dengan interkoneksi dari suatu jaringan. Pada bab ini akan didiskusikan metode-metode *exact* dan metode-metode *heuristic* yang telah dikembangkan untuk menyelesaikan masalah DCMST. Metode-metode *exact* yang telah dikembangkan antara lain *Branch and Bound*, *Lagrangean Relaxation*, dan *Branch and Cut*, sedangkan metode-metode *heuristic* yang telah dikembangkan antara lain metode *greedy* yang berdasarkan algoritma Prim dan Kruskal, *Tabu Search*, *Iterative Refinement*, *Modified Penalty*, dan Algoritma Genetika.

Dalam penulisan buku ini penulis banyak sekali mendapat bantuan. Perkenankanlah penulis mengucapkan terimakasih kepada Shela Malinda, alumni Jurusan Matematika Unila yang sekarang telah bekerja di P.T Great Giant Pineapple atas bantuannya yang telah mendesain cover buku ini.

Penulis menyadari bahwa buku ini sangat jauh dari sempurna, oleh karena itu kritik dan saran dari pembaca sangatlah diharapkan. Penulis mengucapkan terimakasih atas semua kritk dan saran yang ditujukan kepada buku ini.

Bandarlampung, Desember 2022
Penulis

Wamiliana

DAFTAR ISI

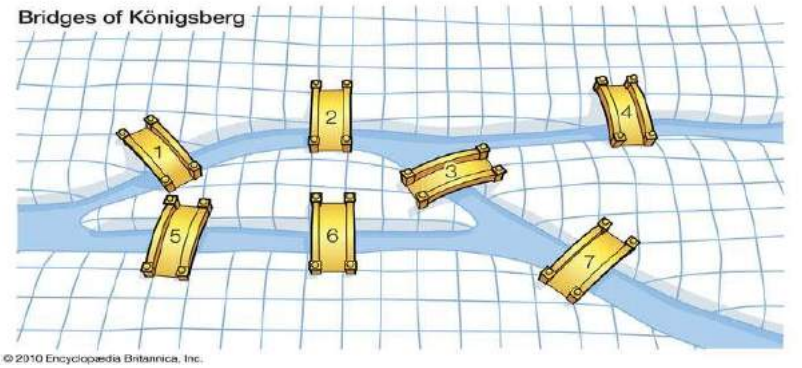
BAB I

KONSEP DASAR TEORI GRAF

Teori graf merupakan salah satu kajian matematika yang memiliki banyak terapan di berbagai bidang sampai saat ini. Kata graf digunakan pada matematika diskrit untuk objek matematika yang terdiri dari himpunan objek-objek dan bagaimana objek-objek tersebut saling berhubungan. Konsep graf digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Representasi visual dari graf adalah menyatakan objek dengan titik/node/simpul/*vertex* dan hubungan antar objek tersebut dinyatakan dengan garis/sisi/*edge*. Struktur graf ini mempunyai terapan yang luas. Sebagai contoh, graf dapat digunakan untuk jaringan transportasi (dengan garis merepresentasikan rute-rute penerbangan yang menghubungkan antar bandara, jalur kereta api yang menghubungkan antar stasiun, jaringan jalan yang menghubungkan antar kota, dll), jaringan listrik, jaringan komputer, jaringan komunikasi, jaringan pipa air, dan lain-lain.

1.1 Sejarah Teori Graf

Sejarah teori graf dimulai sewaktu Euler (1707-1782) pada tahun 1736 memberikan solusi atas masalah jembatan Königsberg. Jembatan Königsberg merupakan tujuh jembatan yang terletak di kota Königsberg. Ketujuh jembatan tersebut menghubungkan dua pulau kecil yaitu pulau Kneiphof dan Lomse yang terletak di tengah sungai Pregel dengan kedua tepi sungai yang digambarkan sebagai berikut:

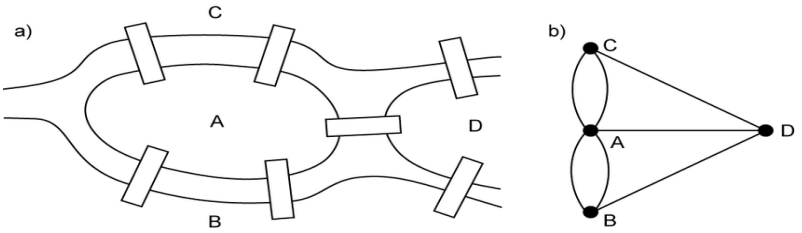


Sumber: https://www.britannica.com/science/Konigsberg-bridge-problem

Gambar 1. Jembatan Königsberg

Permasalahan yang muncul dari jembatan Königsberg tersebut adalah: mungkinkah seseorang, mulai dari mana saja dari empat daratan (yang digambarkan dengan daerah kotak-kotak), melewati tujuh jembatan tersebut tepat sekali dan kemudian kembali ketempat semula? Penduduk kota tersebut mencoba dan ternyata mereka meyakini bahwa hal tersebut adalah tidak mungkin, akan tetapi, mereka tidak dapat memberikan solusi formal atas masalah tersebut.

Euler, dalam tulisannya yang berjudul *Solutio problematis ad geometriam situs pertinentis* merepresentasikan tiap daratan dengan titik, dan setiap jembatan yang menghubungan daratan dengan garis, yang keduanya merupakan unsur utama dari graf. Representasi masalah jembatan Königsberg tersebut dapat dinyatakan dalam Gambar 2 berikut:



Gambar 2a. A, B, C, dan D daratan yang dihubungkan oleh tujuh jembatan

Gambar 2b. Representasi jembatan Konigsberg dalam bentuk titik dan garis.

Euler kemudian memberikan solusi umum atas masalah ini yaitu: tidak mungkin dapat hal tersebut dilakukan kecuali jumlah jembatan (sisi) yang berhubungan (menempel) dengan tiap daratan (titik) adalah genap. Euler memodelkan masalah ini ke dalam graf. Daratan (yang diwakili oleh titik-titik yang disebut juga dengan node/simpul/*vertex*) yang dihubungkan oleh jembatan (yang dinyatakan dengan garis yang disebut juga sebagai sisi /*edge*). Setiap titik diberi label huruf A, B, C, dan D. Jawaban yang dikemukakan oleh Euler adalah: seseorang tidak mungkin melalui ketujuh jembatan itu masing- masing satu kali dan kembali lagi ke tempat asal

keberangkatan jika derajat (*degree*) setiap titik tidak seluruhnya genap.

Derajat suatu titik adalah banyaknya garis yang menempel pada titik tersebut. Dapat dilihat pada Gambar 2b, titik *C* memiliki derajat 3 karena ada tiga garis yang menempel pada titik C, titik *B* dan *D* juga berderajat tiga, sedangkan titik A berderajat 5. Karena semua titik berderajat ganjil, maka tidak mungkin dilakukan perjalananan yang dimulai dari satu titik dan melalui tiap jembatan tepat satu kali dan kembali ke titik awal.

1.2 Istilah-istilah Dasar Teori Graf

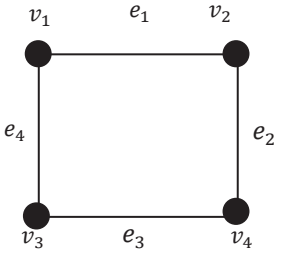
Suatu Graf G (V,E) adalah suatu struktur (V,E) dengan himpunan $V(G) = \{v_1, v_2, v_3, .., v_n\}$ adalah himpunan tak kosong dengan anggota anggotanya disebut titik/node atau *vertex*, sedangkan himpunan $E(G) = \{e_1, e_2, e_3, .., e_n\}$ (yang mungkin kosong) adalah himpunan garis-garis yang menghubungkan titik-titik di himpunan V(G) yang anggotanya disebut garis/sisi atau *edg*e. Notasi lain untuk penulisan garis e_1 adalah e_{ij} atau (v_i,v_j) yaitu garis yang menghubungkan titik v_i dan titik v_j. Anggota himpunan V(G) dapat tak berhingga, dan graf yang dibentuknya merupakan graf tak berhingga. Dalam buku ini, pembicaraan hanya untuk graf dengan banyaknya anggota V(G) yang berhingga.

Suatu graf dengan n titik dan m garis kadang disebut dengan (n,m) graf, sehingga (1,0) graf merupakan graf yang hanya terdiri dari satu titik. Graf yang hanya terdiri dari n titik dan tidak memuat garis disebut dengan graf nol (*null graph*).

Gambar 3. Graf nol dengan empat titik

Setiap graf mempunyai suatu diagram yang berhubungan dengannya. Titik v_i dan v_j disebut menempel (*incidence*) dengan garis e_{ij}, serta titik v_i disebut bertetangga (*adjacent*) dengan titik v_j.

Pada Gambar 4 titik v_1 dan v_2 menempel dengan garis e_1, titik v_1 dan v_3 menempel dengan garis e_4, titik v_3 dan v_4 menempel dengan garis e_3, dan titik v_2 dan v_4 menempel dengan garis e_2. Selain itu, titik v_1 bertetangga dengan titik v_2 dan v_3, titik v_2 bertetangga dengan titik v_1 dan v_4, titik v_3 bertetangga dengan titik v_1 dan v_4, dan titik v_4 bertetangga dengan titik v_2 dan v_3.

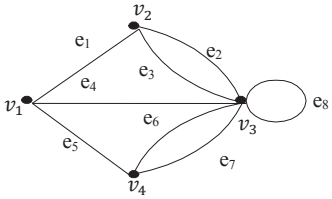


Gambar 4. Contoh graf dengan 4 titik $\{v_1, v_2, v_3, v_4\}$ dan 4 garis $\{e_1, e_2, e_3, e_4\}$.

Garis Paralel dan *Loop*

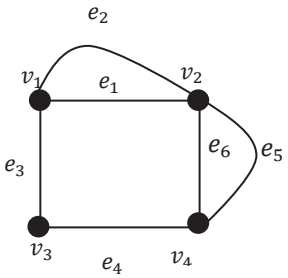
Garis paralel adalah dua garis atau lebih yang menghubungkan pasangan titik yang sama, sedangkan garis yang titik awal dan akhirnya sama disebut dengan *loop*. Suatu graf yang tidak memuat garis paralel atau *loop* disebut dengan

graf sederhana. Graf yang tidak memuat *loop*, akan tetapi memuat garis paralel disebut dengan multigraf, dan graf yang memuat *loop* dan garis paralel disebut dengan *graf semu (pseudo graph)* (Vasudev, 2006).



Gambar 5. Graf yang memuat garis paralel dan *loop*

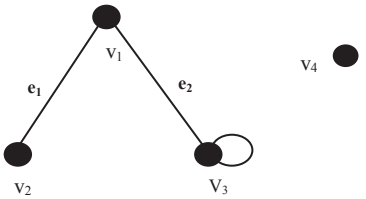
Gambar 4 adalah salah satu contoh graf sederhana, Gambar 5 adalah salah satu contoh graf semu dan Gambar 6 adalah contoh multi graf.



Gambar 6. Salah satu contoh multigraf

Derajat

Derajat (*degree*) dari suatu titik v_i adalah banyaknya garis yang menempel pada titik v_i dan dinotasikan $d(v_i)$. Pada Gambar 7, $d(v_1) = 2$, $d(v_2) = 1$, $d(v_3) = 3$, dan $d(v_4) = 0$. Derajat suatu titik kadang juga disebut dengan *valency* dari titik tersebut.



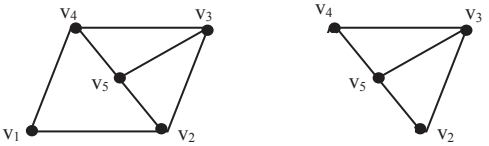
Gambar 7. Contoh graf dengan titik-titik yang berderajat 0, 1, 2 dan 3

Titik terisolasi dan titik *pendant* atau daun (*leaf*)

Suatu titik v_i yang tidak mempunyai garis yang menempel dengannya disebut dengan titik terisolasi. Pada Gambar 7 titik v_4 adalah titik terisolasi. Titik *pendant* yang juga kadang disebut daun (*leaf*) adalah titik yang berderajat satu. Pada Gambar 7, titik v_2 adalah titik *pendant* atau daun.

Subgraf

Graf H dikatakan subgraf dari G jika setiap titik dan garis di H juga merupakan titik dan garis di G, dengan kata lain $V(H) \subseteq V(G)$ dan $E(H) \subseteq E(G)$. H dikatakan *spanning* subgraf dari G jika $V(H) = V(G)$.



Gambar 8. Graf pada Gambar 3b adalah subgraf dari graf pada Gambar 3a

Walk (Perjalanan)

Walk adalah barisan berhingga dari titik/node dan garis/sisi, dimulai dan diakhiri dengan titik sedemikian sehingga setiap garis menempel dengan titik sebelum dan sesudahnya. Tidak ada garis yang muncul lebih dari sekali dalam suatu *walk*.

Contoh *walk* pada Gambar 6 adalah $v_1e_1v_2e_6v_4e_4v_3e_4v_4$.

Lintasan (*Lintasan*)

Lintasan merupakan *walk* yang semua titiknya berbeda. Pada Gambar 6 salah satu contoh lintasan dari graf tersebut adalah $v_1e_1v_2e_6v_4e_4v_3$.

Sirkuit (*circuit*)

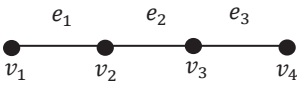
Sirkuit adalah lintasan yang berawal dan berakhir pada titik yang sama. Contoh sirkuit pada Gambar 6 adalah $v_1e_1v_2e_6v_4e_4v_3e_3v_1$. Sirkuit juga kadang disebut dengan siklus (*cycle*).

Graf Terhubung dan Komponen Graf

Graf G dikatakan terhubung jika untuk setiap pasangan titik berbeda di G terdapat lintasan yang menghubungkan kedua titik tersebut. Jika tidak demikian maka G dikatakan tidak terhubung. Komponen graf adalah banyaknya subgraf terhubung pada suatu graf. Gambar 4, 5, dan 6 adalah contoh-contoh graf terhubung, dan Gambar 3 dan 7 adalah contoh-contoh graf tak terhubung. Banyaknya komponen graf pada Gambar 3 adalah 4, sedangkan pada Gambar 7 banyaknya komponen graf adalah 2.

***Incidence* (menempel) *dan adjacent* (bertetangga)**

Jika titik v_i dan v_j adalah titik ujung dari garis e_i maka titik v_i dan v_j disebut *incident* (menempel) dengan garis e_i. Dua garis dikatakan *adjacent* (bertetangga) jika kedua garis tersebut *incident* atau menempel pada titik yang sama. Hal yang sama didefisikan untuk titik. Dua titik dikatakan *adjacent* jika kedua titik tersebut *incident* /menempel pada garis yang sama.

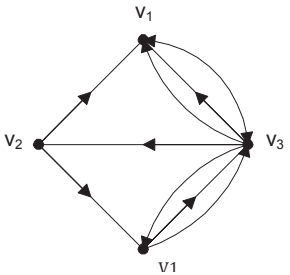


Gambar 9. Contoh *Adjacent* dan *Incident*

Pada Gambar 9 dapat dilihat bahwa v_1 dan v_2 saling bertetangga/*adjacent* karena kedua titik tersebut menempel/*incident* pada garis e_1, dan garis e_1 dan e_2 saling bertetangga/*adjacent* karena kedua titik tersebut menempel/*incident* pada titik v_2.

Graf berarah (*Directed graph*) dan Graf tak-berarah (*Undirected graph*)

Suatu graf yang semua garisnya mempunyai arah disebut graf berarah (*directed graph*). Suatu graf yang semua garisnya tidak mempunyai arah disebut graf tak-berarah (*undirected graph*).

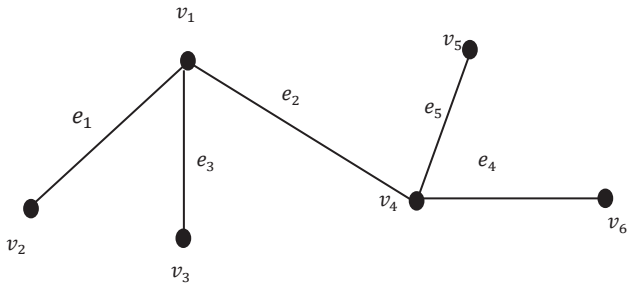


Gambar 10. Contoh graf berarah (*directed graph*).

1.3 *Tree* (Pohon)

Tree (pohon) merupakan salah satu konsep yang penting dalam teori graf, terutama yang berhubungan dengan bentuk-bentuk terapan dari teori graf. *Tree* memainkan peranan yang penting dalam desain dan analisis jaringan. Dalam kenyataannyapun, *tree* umumnya merupakan *backbone* dari jaringan yang terhubung.

Tree merupakan suatu graf terhubung yang tidak mengandung sirkuit. Contoh dari *tree* diberikan dalam graf berikut:



Gambar 11. Contoh *tree*

Seperti telah dijelaskan sebelumnya suatu graf harus mempunyai minimal satu titik, karena itu graf yang hanya terdiri dari titik merupakan *tree* juga, dan disebut dengan *null graph*.

Dari definisi diketahui bahwa suatu *tree* haruslah merupakan graf sederhana, sebab jika graf tersebut tidak sederhana (mengandung garis paralel ataupun *loop*), maka jelas graf tersebut bukan merupakan *tree*.

Karakterisasi dan sifat-sifat dari *tree*

Teorema 1 (Deo, 1982):

Hanya terdapat satu dan hanya satu lintasan (*path*) antara sepasang titik di *tree*.

Bukti :

Karena *tree* T adalah graf terhubung maka pasti ada paling sedikit satu lintasan yang menghubungkan tiap pasang titik di T. Sekarang misalkan ada dua titik di T, katakanlah a dan b yang mempunyai 2 lintasan berbeda yang menghubungkan kedua titik tersebut. Ini berarti kedua lintasan tersebut membentuk suatu sirkuit, dan ini jelas

berarti bahwa T bukan *tree*. Kontradiksi. Maka jelas bahwa lintasan yang menghubungkan titik a dan b harus satu.

Teorema 2. (Deo, 1982):

Jika dalam graf G hanya ada satu dan hanya satu lintasan yang menghubungkan tiap pasangan titik, maka G adalah *tree*.

Bukti:

Karena G memuat hanya satu dan hanya satu lintasan yang menghubungkan tiap pasang titik maka G terhubung. Selain itu, karena hanya terdapat satu dan hanya satu lintasan yang menghubungkan tiap pasang titik di G maka G tidak mengandung sirkuit. Berdasarkan hasil ini (G terhubung dan tidak mengandung sirkuit), maka G adalah *tree*.

Teorema 3. (Deo, 1982):

Suatu *tree* yang mempunyai n titik akan mempunyai n-1 sisi.

Bukti:

Dengan menggunakan induksi matematika.

Dapat dilihat dengan jelas bahwa teorema ini benar untuk n =1.

Misalkan teorema ini benar untuk n = k.

Pertimbangkan suatu *tree* dengan k+1 titik. Misalkan e_k adalah suatu sisi dengan titik-titik ujungnya adalah v_i dan v_j. Sesuai dengan Teorema 1, tidak terdapat lintasan lain yang menghubungkan v_i dan v_j kecuali e_k. Karena itu jika e_k dihilangkan, maka penghilangan ini akan menyebabkan T tidak terhubung. Lebih jauh lagi T – e_k terdiri dari tepat 2 komponen, dan karena tidak terdapat sirkuit pada masing-masing komponen tersebut, jelas bahwa masing-masing komponen tersebut merupakan *tree*.

Berdasarkan induksi, tiap komponen mengandung satu sisi kurangnya dari jumlah titik, ini berarti, T – e_k mempunyai k-1 sisi (dengan jumlah titik k, karena telah dikurangi titik disalah satu ujung e_k). Dengan menggabungkan e_k dengan T maka jumlah titik bertambah satu (menjadi k+1) dan jumlah sisipun bertambah satu (menjadi k) dengan adanya penggabungan e_k tersebut. Jadi jelas bahwa T dengan jumlah titik n akan mempunyai jumlah sisi sebanyak n-1.

Teorema 4. (Deo,1982):
Suatu graf terhubung dengan n titik dan n-1 sisi adalah *tree*
Bukti: (Sebagai latihan)

Salah satu karakteristik dari *tree* adalah jumlah sisi yang menghubungkannya tersebut sudah minimal. Jika suatu graf G dengan n titik mempunyai sisi kurang dari n-1, jelas bahwa graf tersebut tidak terhubung. Sebaliknya, jika graf tersebut mempunyai sisi lebih dari n-1 maka graf tersebut mengandung sirkuit. Ini berarti jumlah sisi yang ada pada suatu *tree* merupakan jumlah sisi minimum agar graf tersebut terhubung.

Suatu graf terhubung dikatakan terhubung minimal (*minimally connected*) jika penghilangan/ pembuangan salah satu sisi dari graf tersebut akan menyebabkan graf menjadi tidak terhubung. Suatu graf yang terhubung minimal tidak akan mungkin mempunyai sirkuit. Sebaliknya, dalam suatu sirkuit dapat dibuang satu sisi dan tetap mempunyai graf yang terhubung.

Teorema 5. (Deo, 1982):

Suatu graf adalah suatu *tree* jika dan hanya jika graf tersebut terhubung minimal.

Bukti: (Sebagai latihan)

Teorema 6. (Deo, 1982):

Suatu graf dengan n titik, n-1 sisi serta tidak mengandung sirkuit, maka graf tersebut terhubung.

Bukti: (Sebagai latihan)

Keenam teorema tersebut merupakan teorema-teorema yang menggambarkan sifat serta karakteristik suatu *tree*.

Pernyataan-pernyataan berikut adalah pernyataan yang berbeda tetapi ekuivalen yang menyatakan sifat serta karakteristik dari suatu *tree* yaitu:

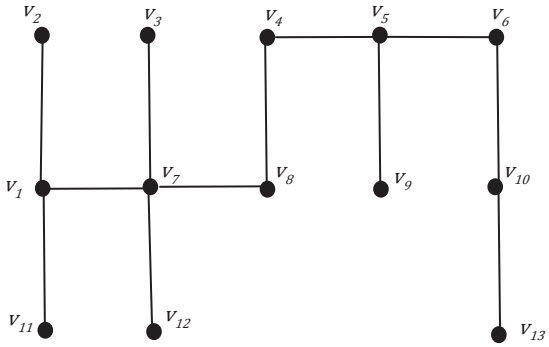
1. G terhubung dan tidak mengandung sirkuit
2. G terhubung dan mempunyai n-1 sisi.
3. G tidak mengandung sirkuit dan mempunyai n-1 sisi.
4. Tepat terdapat satu lintasan antara tiap pasangan titik di G.
5. G terhubung minimal.

Jarak (*Distance*) dan Pusat (*Center*) dari suatu *Tree*

Dalam suatu graf, jarak dari titik s ke titik t, yang dinotasikan dengan d(s,t) adalah panjang dari lintasan terpendek dari s ke t. Jika tidak terdapat lintasan antara s dan t maka jarak tersebut dinotasikan dengan ∞.

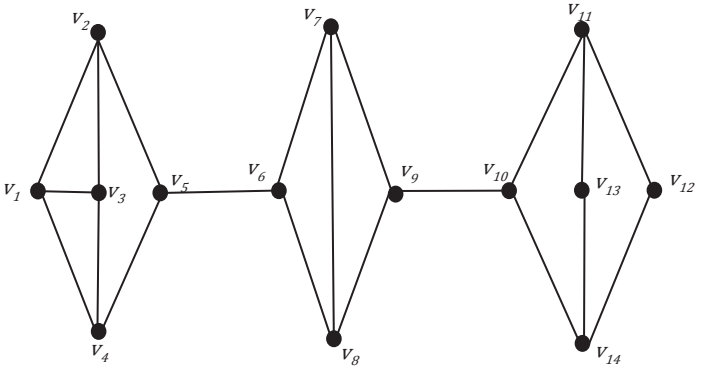
Contoh: Perhatikan *tree* pada Gambar 12.

Pada gambar tersebut jarak antara titik 11 dan 13 adalah 8, sementara jarak titik 12 dan 4 adalah 3. Konsep jarak dalam graf ini berlaku secara umum pada graf, jadi bukan hanya dalam *tree* saja. Sebagai contoh, untuk graf pada Gambar 12 jarak antara titik 1 dan 9 adalah 5, demikian pula jarak antara titik 3 dan 9.



Gambar 12. Salah satu contoh *tree* dengan 11 titik

Pada graf yang bukan merupakan *tree*, dimungkinkan untuk sepasang titik mempunyai lintasan yang berbeda yang menghubungkan keduanya. Untuk itu, dalam penentuan jarak pada graf yang bukan merupakan *tree* harus hati-hati sebab harus dibandingkan dulu yang mana yang paling pendek diantara lintasan-lintasan yang menghubungkan pasangan titik tersebut. Pada *tree*, karena jelas hanya terdapat satu lintasan yang menghubungkan tiap pasangan titik, penentuan jarak antara dua titik lebih mudah dilakukan.

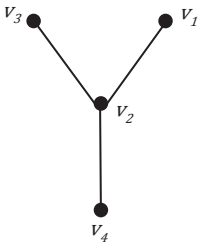


Gambar 13. Salah satu contoh graf dengan 14 titik

Ecentricity E(v) dari titik v dalam graf G adalah jarak dari titik v ke titik yang terjauh dalam G,

$$E(v) = \max_{v_i \in G} d(v, v_i)$$

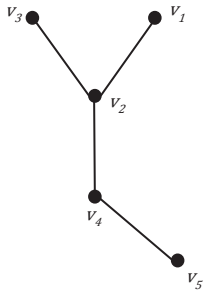
Suatu titik dalam G yang mempunyai *eccentricity* terkecil disebut dengan pusat (*center*) dari graf tersebut.



Gambar 14. Contoh graf dengan satu titik pusat.

Pada graf di Gambar 14, $E(v_1) = 2$, $E(v_2) = 1$, $E(v_3) = 2$, $E(v_4) = 2$. Karena nilai *eccentricity* yang paling kecil terdapat pada titik v_2 maka titik v_2 merupakan pusat dari gaf tersebut.

Dalam suatu graf sangat dimungkinkan untuk mempunyai jumlah pusat yang lebih dari satu. Sebagai contoh, perhatikan graf berikut:



Gambar 15. Contoh graf dengan dua titik pusat.

Pada graf ini, pusat graf terdapat pada titik v_2 dan v_4 yang masing-masing mempunyai *eccentricity* 2. Khusus untuk *tree*, terdapat teorema-teorema berikut:

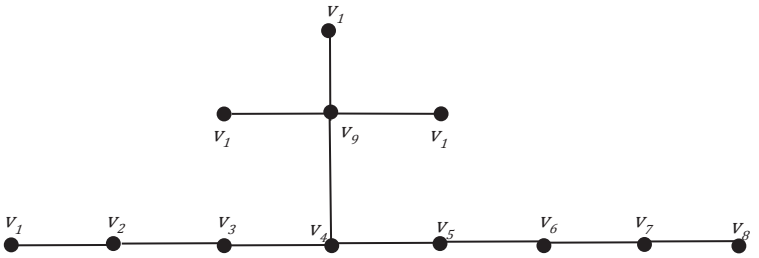
Teorema 3.7 (Deo, 1982) :
Setiap *tree* mempunyai satu atau dua pusat.

Bukti:

Misalkan T_0 adalah *tree* yang mempunyai titik lebih dari dua. Perhatikanlah bahwa jarak maksimum yang dipunyai oleh suatu titik v dalam graf G hanya terjadi jika v adalah *leaf* atau titik *pendant*. Karena T_0 terdiri dari dua titik maka T_0 menpunyai dua atau lebih *leaf*. Buang semua *leaf* dari T_0, hasilnya tetap merupakan *tree* dan sebut T_1. Dengan penghilangan *leaf* ini *eccentricity* dari tiap titik yang tersisa pada T_1 akan berkurang satu. Ini berarti titik yang tadinya merupakan pusat pada T_0, akan tetap merupakan pusat di T_1.

Ulangi lagi proses pembuangan *leaf* dan terbentuk *tree* baru. Ulangi terus langkah ini sampai yang sisa adalah satu titik atau satu sisi. Jika yang tersisa adalah satu titik, berarti titik tersebut yang merupakan pusat dari graf (yang bearti jumlah pusat adalah satu). Jika yang tersisa adalah sisi, maka kedua titik yang merupakan titik ujung dari sisi tersebut merupakan pusat dari graf (yang berarti jumlah pusat graf tersebut ada 2).

Radius dari suatu *tree* adalah *eccentricity* dari pusat *tree* tersebut. Diameter dari suatu *tree* adalah panjang dari lintasan terpanjang yang ada di *tree*.



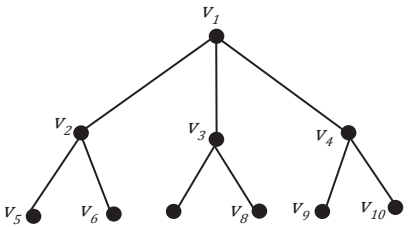
Gambar 16. Contoh graf dengan radius 4 dan diameter 7.

Pada graf ini radius adalah 4 dengan *center* di v_4 dan diameter adalah 7. Perhatikanlah bahwa dalam graf, nilai diameter tidak harus merupakan dua kali radiusnya.

Rooted Tree (Pohon Berakar), *Binary Tree* dan *Spanning tree*

Suatu *tree* yang salah satu titiknya dibedakan dengan titik yang lainnya disebut dengan *rooted tree* (pohon berakar). Titik yang dibedakan tersebut disebut dengan akar atau *root*. Dalam terapannya, *tree* berakar ini dapat merepresentasikan asal mula distribusi suatu barang. Misalkan kota v_1 merupakan

satu-satunya tempat produksi suatu barang. Dari kota v_1 ini kemudian barang tersebut didistribusikan ke kota-kota lain. Dalam hal ini, kota v_1 disebut sebagai *root* (akar) dari *tree* tersebut, darimana semua barang produksi tersebut berasal. Dengan ilustrasi sebagai graf, *rooted tree* tersebut adalah sebagai berikut:



Gambar 17. Contoh *rooted tree*

Secara umum, jika yang ditulis adalah *tree* saja, maka yang dimaksud adalah bentuk *tree* tanpa akar.

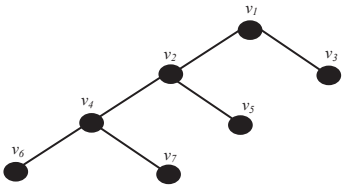
Pohon biner (*binary tree*) adalah suatu pohon yang tepat hanya satu titik yang mempunyai derajat dua sedangkan titik-titik lainnya berderajat satu atau tiga.

Binary tree ini merupakan salah satu bentuk khusus dari *rooted tree*. Bentuk ini sangat banyak digunakan dalam metode *search* (pencarian) dalam ilmu komputer. Pada *binary tree*, titik yang berderajat dua bertindak sebagai *root*. Titik yang bukan merupakan *leaf* dalam suatu binary t*ree* disebut dengan titik dalam (*internal titik*).

Dari dua sifat khusus *binary tree* tersebut, dapat kita ketahui bahwa jumlah titik dalam pada *binary tree* adalah satu kurangnya dari jumlah *leaf*. Pada *binary tree*, titik v dikatakan berada pada tingkat (level) m jika v berada pada jarak sejauh m

dari *root*. *Root* berada pada level 0. Berikut ini diberikan dua sifat khusus dari *binary tree* yaitu:

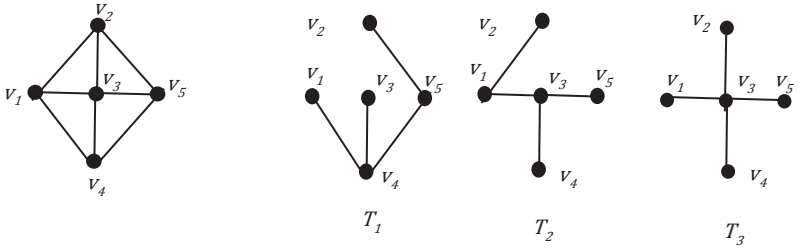
1. Jumlah titik dalam *binary tree* selalu ganjil. Titik yang berderajat dua hanya ada satu dan satu-satunya, sedangkan sisanya sebanyak n-1 lainnya berderajat ganjil.
2. Misalkan p adalah jumlah *leaf* dalam *binary tree*, maka n – p -1 adalah jumlah titik yang berderajat tiga. Akibatnya, jumlah sisi dalam T adalah sebanyak ½ [p + 3 (n-p-1) + 2] = n-1, sehingga p = (n+1)/2



Gambar 18. Contoh *binary tree*

***Spanning tree* (Pohon merentang)**

Spanning tree T dari graf G(V,E) adalah suatu graf terhubung yang memuat semua titik dari G dan tidak memuat sirkuit.



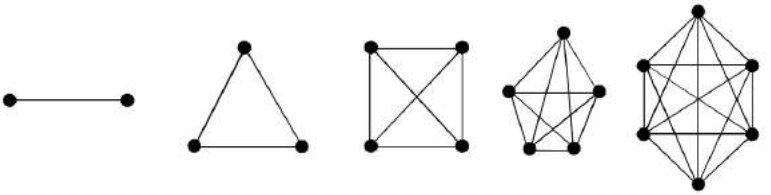
(a) Graf G **(b) Beberapa *spanning* Tree dari G**

Gambar 19. Graf G (a) dan beberapa bentuk *spanning tree* dari G.

Beberapa Bentuk Graf

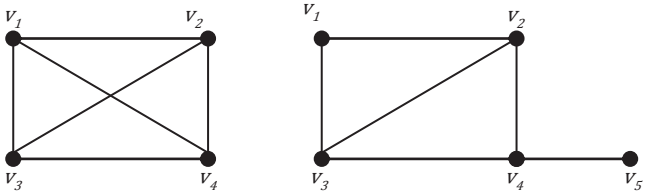
Graf Lengkap

Graf lengkap (*complete graph*) adalah graf yang setiap pasangan titik ada garis yang menghubungkannya. Graf lengkap dengan n titik dinotasikan dengan K_n dan banyaknya garis di K_n adalah ½ n (n-1).



Gambar 20. Graf lengkap untuk n =1,2,3,4,5, dan 6

Clique adalah subgraf yang merupakan graf lengkap.



Gambar 21. (a) Graf K_4 dan (b) adalah graf yang memiliki *clique*

Gambar 19(b) adalah contoh graf dengan 3 *clique* yaitu $\{v_1, v_2, v_3\}$, $\{v_2, v_3, v_4\}$ dan $\{v_4, v_5\}$.

Graf Bipartit

Graf G(V,E) disebut graf bipartit jika V dapat dipartisi menjadi V_1 dan V_2 sehingga $V = V_1 \cup V_2$ dan $V_1 \cap V_2 = \varnothing$ serta untuk setiap $e_{ij} \in E$, $i \in V_1$ dan $j \in V_2$.

Graf Bipartit Lengkap

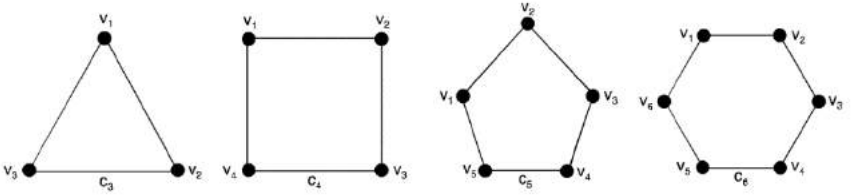
Graf G(V,E) disebut graf bipartit lengkap jika G adalah graf bipartit dan untuk setiap $i \in V_1$ *adjacent* dengan tiap $j \in V_2$. Graf bipartit lengkap dengan $|V_1| = m$ dan $|V_2| = n$ dinotasikan dengan $K_{m,n}$ dan banyaknya garis di $K_{m,n}$ adalah $E(K_{m,n}) = mn$.

Graf Reguler (teratur)

Graf G(V,E) disebut graf r-*reguler* atau r-teratur jika setiap titik di graf berderajat r. Perhatikanlah bahwa setiap *null graph* adalah graf regular berderajat nol, dan graf lengkap K_n adalah graf regular berderajat n-1. Selain itu, jika graf G adalah graf teratur dengan n titik dengan derajat r, maka banyaknya garis adalah (1/2) n r.

Graf Sirkuit (*Cycle*)

Graf sirkuit C_n, $n \geq 3$ adalah graf yang titik awal dan titik akhirnya sama yang terdiri dari n titik v_1, v_2,..., v_n dan garis e_{12}, e_{23}, ..., e_{n-1n}, e_{n1}.



Gambar 22. Graf C_n , n = 3,4,5, dan 6.

BAB II

MINIMUM SPANNING TREE

Minimum Spanning Tree (MST) merupakan salah satu masalah klasik yang muncul sebagai salah satu bentuk terapan dari teori graf. MST umumnya digunakan dalam desain *network*/jaringan. Masalah dasarnya adalah: jika diberikan suatu graf G (V,E) dengan bobot setiap garis e_{ij} adalah c_{ij}, $c_{ij} \geq 0$, maka yang diinginkan adalah mengkonstruksikan atau mendapatkan suatu jaringan yang mempunyai bobot terkecil, dan ini disebut dengan *Minimum Spanning Tree* (MST).

Pada dunia nyata, selain dari bobot terkecil ini ada juga kendala-kendala lain yang harus dipenuhi, misalnya pada jaringan transportasi ada kendala jarak/waktu maksimum untuk delivery suatu produk agar produk tidak rusak; di jaringan radio ada kendala (prasyarat) tambahan, misalnya jumlah maksimal *link* yang diperbolehkan antara stasiun radio.

Minimum spanning tree juga sering digunakan sebagai *backbone* dari masalah-masalah yang muncul dalam desain jaringan dengan menambahkan kendala pada MST. Kendala-kendala yang ditambahkan ke MST tersebut misalnya *degree* (derajat), jarak, *flow*/aliran, keterhubungan (*connectivity*), dan lain sebagainya.

Pada contoh jaringan transportasi di atas, ada kendala jarak yang diberikan agar waktu (yang juga berpengaruh terhadap jarak) untuk mengangkut komoditas tidak melebihi batas yang diberikan. Hal ini dilakukan untuk menjaga agar kualitas dari komoditas yang diangkut tetap baik (jika waktunya lebih dari batas atas waktu, maka komoditas tersebut sudah berkurang kualitasnya), dan sebagainya.

Algoritma untuk menentukan *minimum spanning tree* adalah *polynomial time algorithm*, yang berarti bahwa setiap problem dari MST dapat diselesaikan dengan waktu polynomial sehingga algoritma untuk menyelesaikan MST termasuk kelas P. Akan tetapi, jika selain menentukan MST, terdapat kendala lain yang ditambahkan, misalnya ada penambahan kendala *degree* (*degree constrained minimum spanning tree*), atau diameter (*diameter restricted minimum spanning tree*) seringkali membuat persoalan tersebut menjadi *non deterministic polynomial time* (NP), baik NP-*hard*, maupun NP-*complete*.

Permasalahan umum dari *minimum spanning tree* adalah menentukan garis-garis (*edge*) dari suatu graf yang akan dipilih yang menghubungkan semua titik yang ada pada graf tersebut dengan syarat tidak terbentuk sirkuit dan total bobot dari garis tersebut adalah minimum. Untuk mendapatkan solusi yang diharapkan, akan dipilih garis menurut kriteria optimisasi yang menghasilkan jarak minimum. Jadi, untuk masalah *minimum spanning tree*, yang diinginkan adalah:

1. Graf terhubung;
2. Tidak memuat sirkuit;
3. Total bobot/jarak terkecil (minimum).

2.1. Algoritma Untuk Menentukan *Minimum Spanning Tree*

2.1.1 Algoritma

Algoritma (*algorithm*) merupakan suatu urutan langkah langkah (instruksi) penyelesaian suatu permasalahan yang disusun secara sistematis. Dalam kehidupan sehari hari sebetulnya tanpa disadari kita sering menggunakan algoritma untuk menyelesaikan masalah yang kita hadapi. Sebagai contoh: mengikuti petunjuk dalam suatu resep untuk membuat suatu kue. Konsep algoritma sendiri telah dikenal berabad-abad. Kata algoritma berasal dari kata *algorism* yang merupakan aturan untuk penghitungan yang menggunakan bilangan Hindu-Arab. Kata algorism ini dikaitkan dengan nama ilmuwan pada abad ke 9 yang terkenal yaitu Muhammad ibnu Musa Al Khawaritzmi yang menulis buku dengan judul *Kitab al jabar wal-muqabala* (*The Compendious Book on Calculation by Completion and Balancing*).

Al Khawaritzmi lahir di Khwarizm, Uzbekistan dan meninggal di Baghdad. Al Khawaritzmi juga merupakan ilmuwan yang pertama kali menemukan angka nol dan menuliskannya, yang dengan penemuannya ini membuat matematika dapat berkembang pesat. Karena adanya pengaruh penggunaan kata *arithmetic* untuk penghitungan, maka kata *algorism* kemudian berangsur-angsur berubah menjadi *algorithm*. Kata *algorithm* ini kemudian diserap ke Bahasa Indonesia menjadi algoritma.

Suatu algoritma yang baik mempunyai beberapa karakteristik atau sifat yaitu:

a. Presisi/tepat/pasti: langkah-langkah atau prosedurnya harus dijelaskan atau dijabarkan dengan jelas dan tepat. Setiap algoritma harus memberikan petunjuk yang jelas untuk setiap langkah dan urutan langkah-langkah pun

harus jelas. Detail dari setiap langkah harus dijelaskan, termasuk bagaimana mengatasi jika terjadi kesalahan yang mungkin muncul.

b. Masukan: pada umumnya suatu algoritma memerlukan suatu masukan atau input untuk dijalankan. Masukan adalah besaran nilai yang diberikan kepada algoritma sebelum algoritma tersebut dijalankan. Masukan dari suatu algoritma merupakan data yang akan ditransformasikan sewaktu proses komputasi untuk menghasilkan suatu *output* (luaran). Data yang digunakan sebagai masukan akan sangat berpengaruh terhadap luaran.
c. Luaran: algoritma menghasilkan *output* (luaran). Luaran adalah data atau informasi yang didapat setelah menjalankan langkah-langkah suatu algoritma.
d. Benar: Suatu algoritma harus menghasilkan suatu luaran/*output* yang benar dari suatu masukan/*input* yang diberikan.
e. Berhingga: algoritma akan berhenti setelah beberapa instruksi dilaksanakan. Berhenti disini berarti bahwa kita mendapatkan luaran, atau respon bahwa solusi tidak layak, atau lainnya. Suatu algoritma yang menyebabkan terjadinya *looping* yang terus menerus tanpa berhenti bukanlah suatu algoritma yang baik.
f. Bersifat umum: Suatu algoritma harus dapat diterapkan untuk semua masalah dari bentuk yang diinginkan, bukan hanya dari input tertentu saja.
g. Unik: Langkah lanjutan dari setiap langkah didefinisikan secara tunggal dan tergantung kepada input dari langkah sebelumnya.
h. Efektif: Setiap langkah yang ada pada suatu algoritma harus dapat dilakukan.

Diberikan suatu graf G (V,E), dengan setiap garis e_{ij} diberi bobot c_{ij}, $c_{ij} \geq 0$; *Minimum spanning tree* (MST) dari graf tersebut adalah *spanning tree* dengan jumlah bobot yang minimum. Banyak algoritma yang telah dikembangkan untuk menentukan *minimum spanning tree*, akan tetapi terdapat dua algoritma yang umum dan sering digunakan yaitu Algoritma Prim, dan Algoritma Kruskal. Walaupun demikian, menurut Graham dan Hell (1982) algoritma yang pertama kali digunakan untuk menyelesaikan *minimum spanning tree* digunakan oleh Otakar Boruvka (1899 – 1995) seorang ilmuwan dari Cekoslowakia, pada waktu Perang Dunia pertama, sewaktu ia diminta sahabatnya yang bekerja di fasilitas pembangkit listrik untuk mendesain suatu jaringan distribusi listrik yang efisien. Pada tahun 1926 Boruvka mempublikasikan model/desain tersebut dalam artikelnya *O jistém problému minimálním* (dalam Bahasa Inggris: *On a certain minimal problem*). Selain memberikan desain model yang merupakan suatu MST, Boruvka juga menjabarkan prosedur atau langkah-langkah penyelesaian dari model tersebut. Langkah-langkah tersebut dikenal dengan Algoritma Boruvka, dan merupakan algoritma yang pertama kali dipublikasikan berkaitan dengan penentuan suatu MST dari suatu himpunan kota/pembangkit listrik/tiang-tiang (yang direpresentasikan oleh titik-titik), serta jalan-jalan/kabel-kabel yang menghubungkan kota-kota tersebut. Selain itu, terdapat juga informasi tentang jarak yang menghubungkan antar kota tersebut.

Setelah Algoritma Boruvka dipublikasikan, beberapa algoritma yang mirip dengan Algoritma Boruvka dipublikasikan antara lain oleh Sollin pada tahun 1965. Oleh karena Sollin sering menggunakan algoritma ini dalam penelitian dan artikel yang dipublikasikannya, maka Algoritma Boruvka ini lebih

sering disebut dengan algoritma Sollin. Berikut ini akan diberikan beberapa algoritma untuk menentukan *minimum spanning tree*.

2.1.2 Algoritma Sollin

Konsep awal dari algoritma Sollin atau Boruvka ini adalah menggabungkan setiap *edge* atau garis dengan nilai terendah dari setiap titik di G. Pada tahap awal dari Algoritma Sollin, untuk tiap titik pilih garis dengan bobot terendah yang menempel dengan titik tersebut . Pada Algoritma Sollin ada kemungkinan terbentuknya suatu *forest*.

Langkah-langkah Algoritma Sollin adalah sebagai berikut:

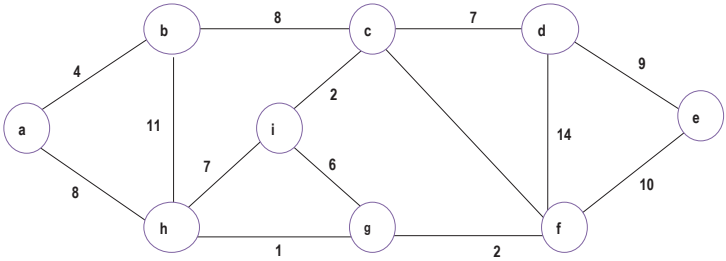
Inisiasi: Graf G dengan n titik dan m garis.

Langkah-langkah:

1. Daftarkan semua titik yang ada di graf G.
2. Untuk setiap titik: pilih garis dengan bobot yang terkecil yang menempel dengan titik tersebut. Jika terjadi ada dua garis dengan bobot terkecil yang sama, pilih satu. (Pada proses ini mungkin saja terbentuk *forest* yang merupakan gabungan dari k komponen dari graf G).
3. Tandai/beri label garis-tersebut dengan ketentuan bahwa garis-garis yang terhubung diberi label/tanda yang sama. Banyaknya label/tanda menunjukkan banyaknya k komponen yang terbentuk.
4. Tentukan garis terkecil yang berada diluar masing-masing komponen.
5. Hubungkan garis terkecil tersebut, yang menghubungkan dua komponen. Untuk dua komponen yang baru terhubung, ganti tanda salah satu komponen menjadi sama dengan komponen lainnya (sehingga jumlah kompenen menjadi k -1).

6. Ulangi langkah 4 sehingga k = 1.
7. Hapus garis-garis yang tidak terpakai
8. Selesai, MST didapat.

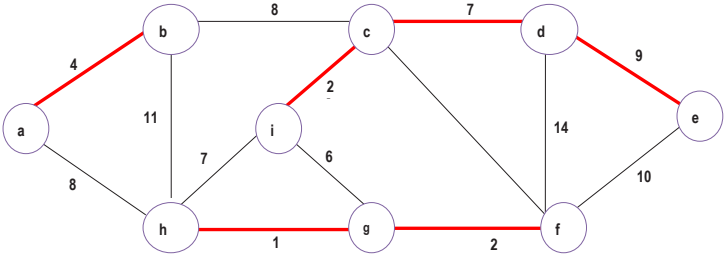
Contoh: Diberikan suatu graf G sebagai berikut:



Gambar 23. Graf G untuk contoh penentuan MST.

1. V = {a, b, c, d, e, f, g, h, i}

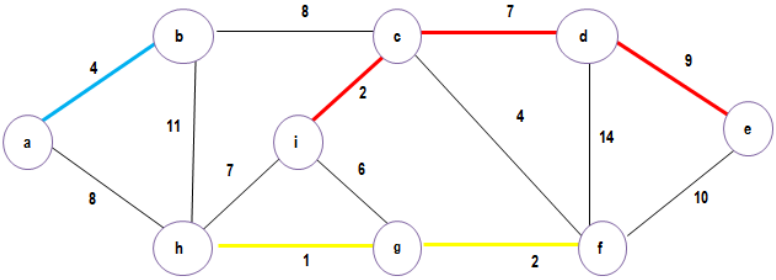
2. Pilih garis dengan bobot terkecil yang menempel pada masing-masing titik.
   Garis yang dipilih adalah garis yang diberi warna merah.



Gambar 24. Graf yang telah diberi tanda garis yang dipilih.

Pada langkah ini, setiap titik telah dihubungkan dengan garis dengan bobot minimal yang menempel pada titik-titik tersebut. Perhatikanlah bahwa pada titik c terdapat dua garis yang menempel. Hal ini terjadi karena garis yang terkecil yang menempel dengan titik c adalah garis dengan bobot 2 yang menghubungkan titik c dengan titik i. Akan tetapi, garis dengan bobot terkecil yang menempel dengan titik d adalah garis dengan bobot 7 yang menghubungkan titik d dengan titik c, sehingga pada titik c menempel dua titik. Banyaknya komponen yang terbentuk adalah 3 komponen.

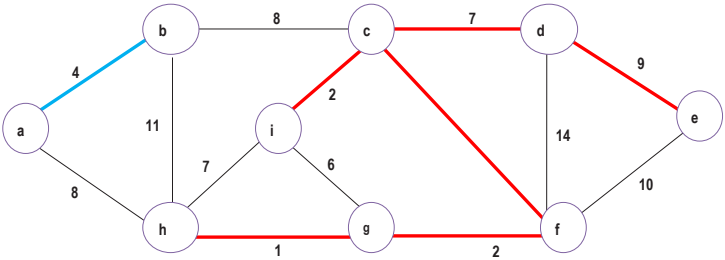
3. Beri tanda masing-masing komponen dengan tanda yang berbeda. Pada contoh ini tiap komponen diberikan tanda berupa warna yang berbeda. Ada 3 komponen yang terbentuk, yaitu komponen yang berwarna hijau, kuning, dan merah.



Gambar 25. Tiga komponen setelah Langkah 3 Algoritma Sollin.

4. Tentukan garis dengan bobot terkecil yang tidak berada pada komponen-komponen yang terbentuk. Garis dengan bobot terkecil yang berada diluar komponen adalah garis yang menghubungkan komponen merah dan komponen kuning dengan bobot 4 (menghubungkan titik c dan f).

5. Hubungkan garis c dan f, dan ganti salah satu tanda komponen yang terhubung sehingga tanda dua komponen yang baru tersebut sama. Pada contoh ini setelah komponen merah dan kuning dihubungkan dengan garis e_{cf} maka warna komponen tersebut dibuat menjadi warna merah (warna kuning diganti dengan warna merah).

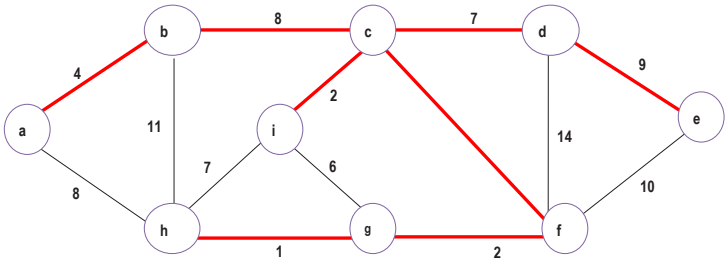


Gambar 26. Proses penggabungan komponen dari 3 menjadi 2.

Banyaknya komponen = 2

6. Ulangi langkah 4 sampai jumlah komponen menjadi 1. Garis dengan bobot terkecil yang menghubungkan kedua komponen (biru dan merah) adalah garis dengan bobot 8 (ada dua garis, satu garis menghubungkan titik a dan h, dan satunya lagi menghubungkan titik b dan c). Karena kedua garis mempunyai bobot sama, maka misalkan kita ambil

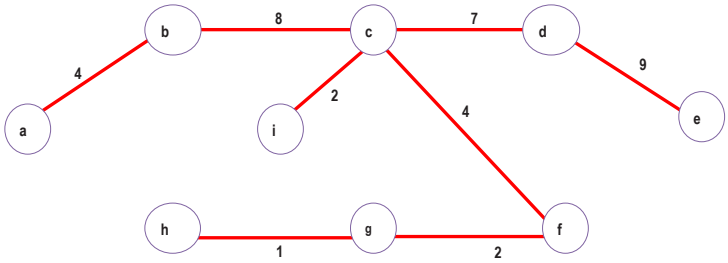
yang menghubungkan titik b dan c. Hubungkan kedua komponen denga garis e_{bc}, dan ganti tanda salah satu komponen menjadi sama dengan tanda komponen lainnya (misal komponen biru kita ganti menjadi komponen merah), sehingga didapat:



Gambar 27. Proses penggabungan komponen dari 2 menjadi satu komponen.

Banyaknya komponen = 1

7. Hapus garis-garis yang tidak terpakai.



Gambar 28. *Minimum Spanning Tree* yang terbentuk dengan menggunakan Algoritma Sollin.

8. Selesai. Total bobot *minimum spanning tree* = 37

Walaupun merupakan algoritma yang pertama kali dikembangkan untuk menyelesaikan MST akan tetapi algoritma Sollin ini lebih jarang digunakan dibandingkan Algoritma Kruskal dan Algoritma Prim.

2.1.3 Algoritma Kruskal

Algoritma Kruskal merupakan salah satu algoritma *greedy* untuk menyelesaikan MST. Pada Algoritma Kruskal, garis-garis pada graf diurut terlebih dahulu berdasarkan bobotnya secara *increasing order* (dari kecil ke besar). Garis yang dimasukkan ke dalam himpunan T adalah garis di graf G sedemikian sehingga T adalah pohon.

Pada keadaan awal, himpunan T berupa himpunan kosong, dan titik-titik di jaringan merupakan *null* graf (graf tanpa garis). Garis dari graf G ditambahkan ke T jika ia tidak membentuk sirkuit di T. Teorema berikut berhubungan dengan algoritma Kruskal.

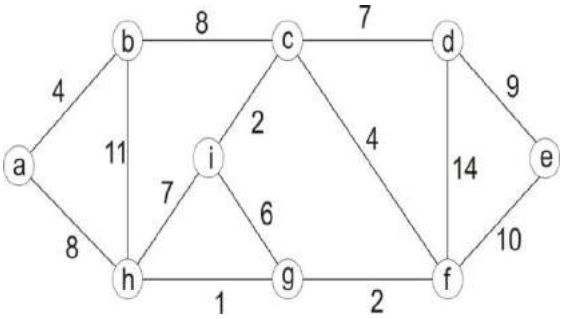
Langkah – langkah algoritma kruskal adalah sebagai berikut:

Set T = ∅, input = graf $G(V, E)$ jumlah titik = n .

1. Sortir garis dari urutan bobot terkecil ke terbesar (*increasing order*).
2. Pilih garis terkecil dalam sortir dan masukkan ke T.
3. Pilih garis berikutnya dalam sortir.
4. Cek apakah penambahan *garis* tersebut pada T menyebabkan terjadinya sirkuit. Jika Ya, buang *garis* tersebut dari sortir dan kembali ke langkah 2. Jika tidak, masukkan *garis* tersebut pada T dan ke langkah 4.
5. Cek apakah |T| - n-1

Jika Ya, STOP, solusi didapat. Jika Tidak, Kembali ke langkah 2.

Contoh: diberikan graf $G(V, E)$ sebagai berikut:



1. Sortir garis dari graf tersebut dengan n = 9 (banyaknya titik)
   $e_{hg} = 1$
   $e_{ic} = 2$
   $e_{fg} = 2$
   $e_{ab} = 4$
   $e_{cf} = 4$
   $e_{ig} = 6$
   $e_{cd} = 7$
   $e_{ih} = 7$
   $e_{ah} = 8$
   $e_{bc} = 8$
   $e_{de} = 9$
   $e_{ef} = 10$
   $e_{bh} = 11$
   $e_{df} = 14$
2. Pilih $e_{hg} = 1$, $T = \{e_{hg}\}$, $|T| = 1$
3. Pilih $e_{ic} = 1$
4. Penambahan e_{ic} di T tidak menyebabkan terbentuknya sirkuit, jadi $T = \{e_{hg}, e_{ic}\}, |T| = 2$.
5. $|T| = 2 , \neq 9 - 1$. Kembali ke langkah 2.

Pilih e_{fg}.
Penambahan e_{fg} di T tidak menyebabkan terbentuknya sirkuit, jadi T = $\{e_{hg}, e_{ic}, e_{fg}\}$, |T| = 3.
|T| = 3, 3 ≠8. Kembali ke langkah 2

Pilih e_{ab}=4.
Penambahan e_{ic} di T tidak menyebabkan terbentuknya sirkuit, jadi T=$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}\}$, |T| = 4.
|T| = 4, 4≠8. Kembali ke langkah 2

Pilih e_{cf} = 4.
Penambahan e_{cf} di T tidak menyebabkan terbentuknya sirkuit, jadi T = $\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}\}$, |T| = 5.
|T| = 5, 5≠8. Kembali ke langkah 2

Pilih e_{ig} = 6.
Penambahan e_{ig} di T menyebabkan terbentuknya sirkuit, sehingga e_{ig} dibuang. Kembali ke langkah 2

Pilih e_{cd} = 7.
Penambahan e_{cd} di T tidak menyebabkan terbentuknya Sirkuit, jadi T=$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}, e_{cd}\}$, |T| = 6.
|T| = 6 , 6 ≠ 8. Kembali ke langkah 2

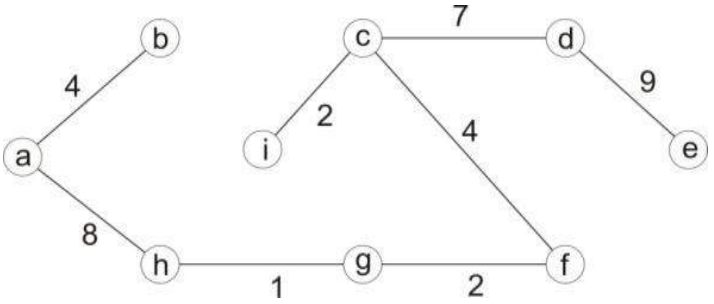
Pilih e_{ih} = 7.
Penambahan e_{ih} di T menyebabkan terbentuknya sirkuit, sehingga e_{ih} dibuang. Kembali ke langkah 2

Pilih e_{ah} = 8.
Penambahan e_{ah} di T tidak menyebabkan terbentuknya sirkuit, T=$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}, e_{cd}, e_{ah}\}$, |T| = 7.
|T| = 7, 7 ≠ 8. Kembali ke langkah 2

Pilih $e_{bc} = 8$.
Penambahan e_{bc} di T menyebabkan terbentuknya sirkuit, sehingga e_{bc} dibuang. Kembali ke langkah 2

Pilih $e_{de} = 9$.
Penambahan e_{de} di T tidak menyebabkan terbentuknya sirkuit
$T=\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}, e_{cd}, e_{ah}, e_{de}\}$, $|T| = 8$
$|T| = 8$ STOP.

Karena banyaknya elemen di T sama dengan n=1, maka telah terbentuk *spanning tree* T, yang juga sekaligus *Minimum Spanning Tree*. *Minimum Spanning Tree* yang terbentuk diberikan pada Gambar 29 dengan total bobot MST = 37



Gambar 29. *Minimum Spanning Tree* yang terbentuk dengan menggunakan Algoritma Kruskal.

Proses algoritma Kruskal ini dapat juga dilakukan dengan menggunakan tabel sebagai berikut:

Tabel 1. Prosedur Algortima Kruskal.

Iterasi	Garis yang dipilih	Membentuk *cycle* atau sirkuit?		T	|T| = n -1 ?		Keterangan
		Ya	Tidak		Ya	Tidak	
1	e_{hg}		√	$\{e_{hg}\}$		√	
2	e_{ic}		√	$\{e_{hg}, e_{ic}\}$		√	
3	e_{fg}		√	$\{e_{hg}, e_{ic}, e_{fg}\}$		√	
4	e_{ab}		√	$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}\}$		√	
5	e_{cf}		√	$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}\}$		√	
6	e_{ig}	√		$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}\}$		√	Garis e_{ig} dibuang dari list.Elemen di T tetap
7	e_{cd}		√	$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}, e_{cd}\}$		√	
8	e_{ih}	√		$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}, e_{cd}\}$		√	Garis e_{ih} dibuang dari list. Elemen di T tetap
9	e_{ah}		√	$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}, e_{cd}, e_{ah}\}$			
10	e_{bc}	√		$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}, e_{cd}, e_{ah}\}$		√	Garis e_{bc} dibuang dari list. Elemen di T tetap
11	e_{de}		√	$\{e_{hg}, e_{ic}, e_{fg}, e_{ab}, e_{cf}, e_{cd}, e_{ah}, e_{de}\}$	√		Karena |T| = n -1 = 8, maka STOP.
MST yang terbentuk dapat dikonstruksi dengan menggambarkan *tree* sesuai dengan urutan garis yang masuk ke dalam T. Dari konstruksi ini diketahui bahwa pada proses pembentukan MST dengan Algoritma Kruskal dimungkinkan terbentuknya *forest* (hutan).							

2.2.3 Algoritma Prim

Algoritma ini dikembangkan pertama kali oleh Vojtěch Jarník pada tahun 1930 dan kemudian dikembangkan dan dipublikasikan kembali oleh R. C. Prim pada tahun 1957, dan E. W. Dijsktra pada tahun 1959, akan tetapi penamaan algoritma ini dengan Algoritma Prim lebih dikenal secara luas.

Untuk menentukan MST dari graf G dengan Algoritma Prim mula-mula dipilih satu titik sembarang (misal v_1). Kemudian tambahkan satu garis yang berhubungan dengan v_1 dengan bobot yang paling minimum/kecil (misal e_1) dan ujung lainnya ke T sehingga T terdiri dari garis e_1 dan 2 buah titik-titik ujung garis e_1 (salah satunya adalah v_1). Pada setiap langkah selanjutnya pilih satu garis dalam E(G) yang bukan anggota E(T) dengan sifat:

a. Garis tersebut mempunyai nilai yang minimum
b. Garis tersEbut berhubungan dengan salah satu titik di V(T)
c. Langkah tersebut diulang-ulang hingga memperoleh (n-1) garis dalam E(T) (n adalah jumlah titik pada G).

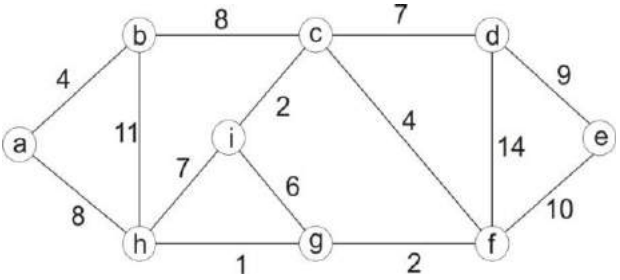
Inisiasi: $T = \emptyset$, $V = \emptyset$.
Langkah –langkah:

1. Tentukan salah satu titik sebagai *root*.
2. Masukkan *root* ke V.
3. Tentukan garis minimum yang terhubung dengan *root*, pilih dan masukkan ke T. Titik ujung garis tersebut masukkan ke V.
4. Pilih garis minimum yang terhubung dengan titik – titik di V dan cek apakah membentuk sirkuit jika ditambahkan ke T.
   Jika ya, maka buang garis tersebut dan pilih lagi garis minimum berikutnya.

Jika tidak, masukkan garis nya di T dan titik nya di V.

5. Cek T = n − 1 ?
   Jika ya STOP.
   Jika tidak, ulangi langkah 4.

Contoh: Diberikan graf terhubung sebagai berikut:



Algoritma Prim

$T = \emptyset,\ V = \emptyset$

1. Pilih a sebagai *root*.
2. V = {a}
3. Garis yang dipertimbangkan: $e_{ab} = 4$ dan $e_{ah} = 8$. Pilih $e_{ab} = 4 \rightarrow T = \{e_{ab}\}, V = \{a, b\}$
4. Garis yang dipertimbangkan: $e_{ah} = 8$, $e_{bc} = 8$, dan $e_{bh} = 11$. Pilih $e_{bc} = 8$.
   Penambahan e_{bc} di T tidak menyebabkan terbentuknya sirkuit. $T = \{e_{ab}, e_{bc}\}$, $V = \{a, b, c\}$
5. $|T| = 2 \neq 8$. Ulangi Langkah 4.
   Garis yang dipertimbangkan:
   e_{ah}=8, e_{bh}=11, e_{ci}=2, e_{cf} =4 dan e_{cd} =7
   Pilih e_{ci} = 2.
   Penambahan e_{ci} di T tidak menyebabkan terbentuknya sirkuit. T ={e_{ab}, e_{bc}, e_{ci}}, V ={a, b, c, i}
   $|T| = 3 \neq 8$. Ulangi Langkah 4.

Garis yang dipertimbangkan:
e_{ah}=8, e_{bh}=11, e_{ci}=2, e_{cf} =4 dan e_{cd} =7, e_{ig}=6, dan e_{ih}=7.

Pilih e_{cf} = 4.

Penambahan e_{cf} di T tidak menyebabkan terbentuknya sirkuit. $T = \{e_{ab}, e_{bc}, e_{ci}, e_{cf}\}$, V = {a, b, c, i, f}.

|T| = 4 ≠ 8 . Ulangi Langkah 4.

Garis yang dipertimbangkan:

$e_{ah} = 8$, $e_{bh} = 11$, $e_{cd} = 7$, $e_{ig} = 6$, $e_{ih} = 7$, $e_{fg} = 2$, $e_{fd} = 14$, $e_{fe} = 10$.

Pilih $e_{fg} = 2$

Penambahan e_{fg} di T tidak menyebabkan terbentuknya sirkuit. $T = \{e_{ab}, e_{bc}, e_{ci}, e_{cf}, e_{fg}\}$, V = {a, b, c, i, f, g}.

|T| = 5 ≠ 8. Ulangi Langkah 4.

Garis yang dipertimbangkan:

e_{ah}= 8, e_{bh}=11, e_{cd}= 7, e_{ig} = 6, e_{ih}=7, e_{fd}=14, e_{fe}=10, e_{gh}=1.

Pilih e_{gh}=1.

Penambahan e_{gh} di T tidak menyebabkan terbentuknya sirkuit. T={e_{ab}, e_{bc}, e_{ci}, e_{cf}, e_{fg}, e_{gh}}, V= {a, b, c, i, f, g, h}.

|T| = 6 ≠ 8 . Ulangi Langkah 4.

Garis yang dipertimbangkan:

e_{ah}= 8, e_{bh}=11, e_{cd}= 7, e_{ig} = 6, e_{ih}=7, e_{fd}=14, e_{fe}=10.

Pilih e_{ig}= 6.

Penambahan e_{ig} di T menyebabkan terbentuknya sirkuit, buang e_{ig} dari daftar. Ulangi Langkah 4.

Garis yang dipertimbangkan:

e_{ah}= 8, e_{bh}=11, e_{cd}= 7, e_{ih}=7, e_{fd}=14, e_{fe}=10.

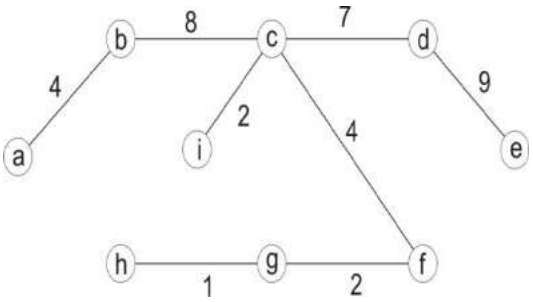
Pilih $e_{cd} = 7$.
Penambahan e_{cd} di T tidak menyebabkan terbentuknya sirkuit. T={e_{ab}, e_{bc}, e_{ci}, e_{cf}, e_{fg}, e_{gh}, e_{cd}},
V= {a, b, c, i, f, g, h, d}.
$|T| = 7 \neq 8$. Ulangi Langkah 4.
Garis yang dipertimbangkan:
e_{ah}= 8, e_{bh}=11, e_{ih}=7, e_{fd}=14, e_{fe}=10, e_{de}=9.

Pilih $e_{ih} = 7$.
Penambahan e_{ih} di T menyebabkan terbentuknya sirkuit, buang e_{ih} dari daftar. Ulangi Langkah 4.
Garis yang dipertimbangkan:
e_{ah}= 8, e_{bh}=11, e_{fd}=14, e_{fe}=10, e_{de}=9.

Pilih $e_{ah} = 8$.
Penambahan e_{ah} di T menyebabkan terbentuknya sirkuit, buang e_{ah} dari daftar. Ulangi Langkah 4.
Garis yang dipertimbangkan:
e_{bh}=11, e_{fd}=14, e_{fe}=10, e_{de}=9.

Pilih e_{de}=9.
Penambahan e_{de} di T tidak menyebabkan terbentuknya sirkuit. T={e_{ab}, e_{bc}, e_{ci}, e_{cf}, e_{fg}, e_{gh}, e_{cd}, e_{de}},
V= {a, b, c, i, f, g, h, d, e}.
$|T| = 8$ STOP.

Karena banyaknya elemen di T sama dengan n=1, maka telah terbentuk *spanning tree* T, yang juga sekaligus *Minimum Spanning Tree.*



Gambar 30. *Minimum Spanning Tree* yang terbentuk dengan menggunakan Algoritma Prim

Bobot total MST = 37

Proses menentukan MST dengan menggunakan Algoritma Prim ini dapat juga dilakukan dengan menggunakan tabel sebagai berikut:
Misal: ambil *root* = a

Tabel 2. Prosedur Algortima Prim

Iterasi	Garis yang dipertimbangkan	Garis yang dipilih	Membentuk *cycle* atau sirkuit?		V	T	|T| = n -1?		Keterangan
			Ya	Tidak			Ya	Tidak	
1	e_{ab}=4 e_{ah}=8	e_{ab} =4		√	{a,b }	{e_{ab}}		√	
2	e_{ah}=8 e_{bc}=8 e_{bh}=11	e_{bc}= 8		√	{a,b,c}	{e_{ab}, e_{bc}}		√	
3	e_{ah}=8, e_{bh}=11 e_{ci}=2, e_{cf}=4, e_{cd}=7	e_{ci}= 2		√	{a,b, c,i}	{e_{ab},e_{bc},e_{ci}}		√	
4	e_{ah}=8, e_{bh}=11 e_{cf}=4, e_{cd}=7 e_{ig}=6, e_{ih}=7	e_{cf}= 4		√	{a,b,c,i,f}	{e_{ab}, e_{bc}, e_{ci}, e_{cf} }		√	
5	e_{ah}=8, e_{bh}=11 e_{cd}=7, e_{ig}=6 e_{ih}=7, e_{fd}=14	e_{fg}=2		√	{a,b,c,i,f,g}	{e_{ab},e_{bc}, e_{ci}, e_{cf}, e_{fg}}		√	.

Iterasi	Garis yang dipertimbangkan	Garis yang dipilih	Membentuk *cycle* atau sirkuit?		V	T	|T| = n -1?		Keterangan
			Ya	Tidak			Ya	Tidak	
	e_{fg}=2, e_{fe}=10								
6	e_{ah}= 8, e_{bh}= 11 e_{cd}=7, e_{ig}=6 e_{ih}=7, e_{fd}=14 e_{fe}=10, e_{gh}=1	e_{gh}=1		√	{a,b,c,i,f,g, h}	{e_{ab},e_{bc}, e_{ci}, e_{cf}, e_{fg}, e_{gh}}		√	.
7	e_{ah}= 8, e_{bh}= 11 e_{cd}=7, e_{ig}=6 e_{ih}=7, e_{fd}=14 e_{fe}=10	e_{ig}= 6	√		{a,b,c,i,f,g, h}	{e_{ab},e_{bc}, e_{ci}, e_{cf}, e_{fg}, e_{gh}}		√	Garis e_{ig} dibuang dari list, T dan V tetap.
8	e_{ah}=8, e_{bh}=11 e_{cd}=7,e_{ih}=7 e_{fd}=14, e_{fe}=10	e_{cd}=7		√	{a,b,c,i,f,g, h,d}	{e_{ab},e_{bc}, e_{ci}, e_{cf}, e_{fg}, e_{gh}, e_{cd}}		√	
9	e_{ah}=8, e_{bh}=11 e_{ih}=7, e_{fd}=14 e_{fe}=10, e_{de}=9	e_{ih}=7	√		{a,b,c,i, f,g, h,d}	{e_{ab},e_{bc},e_{ci}, e_{cf},e_{fg}, e_{gh}, e_{cd}}		√	Garis e_{ih} dibuang dari list, T dan V tetap

Iterasi	Garis yang dipertimbangkan	Garis yang dipilih	Membentuk *cycle* atau sirkuit?		V	T	|T| = n -1?		Keterangan
			Ya	Tidak			Ya	Tidak	
10	e_{ah}= 8, e_{bh}=11 e_{fd}=14, e_{fe}=10 e_{de}=9	e_{ah}=8	√		{a,b,c,i,f,g, h,d}	{e_{ab},e_{bc},e_{ci}, e_{cf}, e_{fg}, e_{gh}, e_{cd}}		√	Garis e_{ah} dibuang dari list, T dan V tetap
11	e_{bh}=11, e_{fd}=14 e_{fe}=10, e_{de}=9	e_{de}=9		√	{a,b,c,i,f,g, h,d,e}	{e_{ab},e_{bc}, e_{ci}, e_{cf}, e_{fg},e_{gh}, e_{cd}, e_{de}}	√		Karena |T|= n-1= 8, maka STOP.

Catatan:

1. Pada Algoritma Sollin dan Kruskal dimungkinkan terjadinya *forest* pada waktu proses pembentukan *spanning tree*, akan tetapi pada Algoritma Prim dari awal hingga selesai yang terbentuk adalah *tree*.
2. Pemilihan "*root*" pada Algoritma Prim dapat dilakukan atau dimulai dari titik mana saja (sebarang).
3. Pada akhir pembentukan *spanning tree*, MST yang terbentuk dari Algoritma Kruskal maupun Prim dapat saja berbeda, tetapi nilai total dari bobot harus sama.

2.2.4 Kompleksitas Algoritma

Suatu algoritma yang telah dikembangkan diharapkan selain mempunyai luaran yang benar dan sesuai dengan yang diinginkan, juga harus efisien. Kebenaran suatu algoritma harus diuji dengan memberikan masukan tertentu dan melihat kinerja algoritma tersebut. Kinerja algoritma dalam hal ini adalah waktu yang diperlukan untuk melalukan langkah-langkah algoritma tersebut dan juga berapa banyak memori yang diperlukan. Sehingga, suatu algoritma yang baik atau efisien adalah algoritma yang selain memberikan luaran yang benar juga menggunakan waktu serta ruang memori yang minimum.

Ada dua macam kompleksitas algoritma, yaitu kompleksitas waktu (*time complexity*) dan kompleksitas ruang (*space complexity*). Kompleksitas waktu adalah jumlah tahap komputasi yang diperlukan untuk menjalankan suatu algoritma yang dianggap sebagai fungsi dengan ukuran masukan/input n, sedangkan kompleksitas ruang adalah jumlah memori yang diperlukan oleh struktur data yang ada dalam algoritma sebagai fungsi dengan ukuran masukan/input n.

Kompleksitas ruang dan kompleksitas waktu sangat tergantung kepada banyak hal, antara lain perangkat keras, sistem operasi, prosesor, dan lain-lain. Walaupun demikian, dalam hal ini kita tidak mempertimbangkan faktor-faktor tersebut sewaktu menganalisis suatu algoritma karena suatu algoritma yang dijalankan di komputer yang berbeda akan berbeda juga kecepatan waktu yang diperlukan untuk menyelesaikan algoritma itu. Oleh karena itu, sewaktu menghitung kompleksitas suatu algoritma yang digunakan adalah memandang suatu algoritma sebagai fungsi dengan

nilai masukan n, dan yang akan kita diskusikan disini adalah kompleksitas waktu dari algoritma.

Misalkan n menyatakan besar input suatu algoritma dan T(n) adalah kompleksitas waktu yang menyatakan jumlah operasi yang dilakukan untuk melaksanakan algoritma sebagai fungsi dari input/masukan n. Masukan n dapat menyatakan banyaknya elemen dalam suatu baris atau lainnya, tergantung masalah apa yang diselesaikan dengan algoritma tersebut. Sebagai contoh, jika algoritma yang dijalankan adalah algoritma untuk melakukan sortir/pengurutan bilangan, maka n adalah jumlah elemen baris, sedangkan dalam algoritma perkalian matriks maka n menyatakan ukuran matriks n x n. Selain itu, ada juga yang menggunakan besaran input selain n, misalnya jika masukan algoritma adalah graf maka ukuran masukan adalah n (jumlah titik) dan m (jumlah sisi).

Secara ideal, untuk menghitung kompleksitas suatu algoritma maka kita menghitung semua operasi yang ada dalam algoritma tersebut. Akan tetapi, secara umum, penghitungan kompleksitas suatu algoritma dapat diwakili oleh jumlah operasi abstrak yang mendasari suatu algoritma yang disebut dengan *basic operation* (operasi dasar). Sebagai contoh, pada algoritma untuk mendapatkan bilangan terbesar dari n bilangan yang didaftarkan secara acak, maka bilangan pertama yang ada pada daftar diberikan nilai maksimum sementara. Kemudian, perbandingan dilakukan terhadap bilangan pertama dan kedua untuk menentukan mana yang lebih besar, dan meng *update* nilai maksimum dari dua nilai yang dibandingkan. Proses ini dilanjutkan untuk setiap bilangan yang ada pada daftar ($i \leq n$). Sehingga, operasi dasar yang diperlukan adalah sebanyak (n-1) untuk mengecek apakah sudah sampai pada bilangan terakhir pada daftar, (n-1)

untuk membandingkan mana yang lebih besar dari dua bilangan, serta 1 operasi untuk keluar dari *loop* i, ketika i = n+1. Total operasi dasar yang diperlukan adalah sebanyak 2 (n-1) + 1 = 2n -1. Jadi, kompleksitas algoritma mencari bilangan terbesar adalah O(n).

Definisi :

T(n) = O (f(n)) jika ada konstanta C dan n_0 sehingga T(n) ≤ C f(n) untuk n ≥ n_0.

T(n) = O (f(n)) dibaca sebagai T(n) adalah O(f(n)) yang berarti T(n) berorde paling besar n.

Dalam ilmu komputer, notasi O-besar (*Big Oh*) digunakan untuk menganalisis dan mengklasifikasi efisiensi dari suatu algoritma sewaktu ukuran dari masalah menjadi besar. Notasi O-besar mencirikan atau mengkarakterisasi fungsi berdasarkan tingkat pertumbuhannya: fungsi yang berbeda dengan tingkat pertumbuhan yang sama dapat direpresentasikan menggunakan notasi O yang sama. Huruf O digunakan karena tingkat pertumbuhan suatu fungsi juga dirujuk sebagai *orde* dari suatu fungsi.

Sebagai contoh, waktu (jumlah langkah) yang diperlukan untuk mengeksekusi suatu algoritma dengan ukuran n adalah $T(n) = 10n^2 - n + 1$. Untuk n yang besar, suku yang memuat n^2 akan mendominasi nilai yang didapat sehingga suku-suku yang lain dapat diabaikan, misalnya untuk n = 100 maka suku yang memuat n^2 akan menjadi 1000 kali lebih besar dari suku yang memuat n. Sehingga, suku-suku yang didominasi tersebut dapat diabaikan pada nilai ekspresi. Lebih jauh, koefisien menjadi tidak relevan jika dibandingkan dengan ekspresi lainnya seperti ekspresi yang memuat n^3 atau n^4.

BAB III

OPTIMISASI KOMBINATORIAL

3.1 Optimisasi Kombinatorial.

Optimisasi kombinatorial merupakan salah satu bidang dari matematika terapan yang sangat berkembang yang menggabungkan teknik-teknik dari kombinatorial, pemograman linear serta menggunakan teori komputasi dan komputer untuk mengoptimalkan masalah masalah dengan struktur diskrit. Dalam terapannya, optimisasi kombinatorial muncul dalam berbagai masalah antara lain masalah transportasi, telekomunikasi, jaringan komputer, penjadwalan, perencanaan, distribusi, dan lain-lain.

Secara struktur, optimasi kombinatorial merupakan subbidang dari optimasi matematika yang tujuannya adalah menentukan solusi yang optimal, di mana himpunan solusi layak merupakan variabel diskrit atau himpunan yang anggotanya merupakan variabel-variabel diskrit. Optimasi kombinatorial terkait dengan riset operasi, teori algoritma, dan teori kompleksitas komputasi. Masalah optimasi kombinatorial yang umum adalah *Travelling Salesman Problem* (TSP), masalah *Minimum Spanning Tree* (MST), masalah *vertex cover*, *set cover*, *knapsack*, dan lain-lain

Dalam optimisasi kombinatorial suatu masalah umumnya dapat dimodelkan dengan menggunakan konsep teori graf dengan merepresentasikan masalah tersebut dalam bentuk titik dan garis. Titik dapat mewakili kota/komputer/bandara/ stasiun dan lain sebagainya, sedangkan garis dapat mewakili jalan/kabel/route penerbangan/jalur kereta api dan lain sebagainya. Sedangkan untuk menyatakan hubungan nonstruktural dapat dengan memberikan bobot pada garis. Bobot pada garis ini dapat mewakili jarak/waktu/biaya dan lain sebagainya.

Graf digunakan sebagai alat untuk merepresentasi suatu masalah karena sifat graf tersebut yang fleksibel. Secara umum tidak ada ketentuan bagaimana cara menggambar garis pada graf (kecuali graf planar, dimana dalam menggambarkan graf tersebut pada bidang, tidak boleh ada garis yang saling berpotongan). Garis dapat digambarkan berbentuk garis lurus, kurva, atau lainnya.

Salah satu tantangan dalam optimisasi kombinatorial adalah mengembangkan suatu algoritma yang efisien untuk menyelesaikan suatu masalah. Metode-metode yang saat ini diterima dengan baik untuk menyelesaikan masalah optimisasi kombinatorial, tiga atau empat dekade yang lalu hampir tidak dianggap serius dapat menyelesaikan masalah tersebut, karena hampir tidak ada yang dapat melakukan perhitungan yang akan dilakukan. Akan tetapi, dengan perkembangan teknologi informasi dan komputer yang demikian pesat, maka masalah yang dulunya dianggap sulit dilakukan, dapat diselesaikan dalam waktu yang relatif cepat.

Model optimisasi kombinatorial sering juga disebut dengan model pemograman bilangan bulat (*integer programming*) atau model IP. Dalam model IP ini beberapa

atau semua variabel harus bernilai diskrit, dan umumnya pada banyak kasus nilai-nilai ini merupakan bilangan bulat. Oleh karena itu, model ini disebut dengan pemodelan bilangan bulat (*integer programming*). Secara umum, bentuk model *integer programming* (IP) adalah :

Obj : Min $\mathbf{c^T x}$
Kendala $\mathbf{Ax} \leq \mathbf{b}$
$x_j \in \mathbb{Z}^n$, j =1, 2, ..., n

$\mathbb{Z}^n$ adalah himpunan integer n – vektor, $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ adalah n-vektor bernilai bilangan bulat, dan **c** adalah n-vektor, **A** adalah matriks ukuran m x n, m adalah banyaknya kendala pertidaksamaan, dan **b** adalah m – vektor. Jika sebagian variabel x_i diperbolehkan bernilai real, maka persoalan tersebut menjadi persoalan campuran pemograman bilangan bulat (*Mixed Integer Programming problem*) atau MIP, jika semua nilai x_i mengambil nilai 0 atau 1, maka masalah tersebut menjadi 0 – 1 *Integer Programming* (0 -1 IP).

Menyelesaikan masalah IP tidak semudah menyelesaikan masalah Program Linear (PL). Hal ini karena jika pada masalah PL daerah layak merupakan suatu himpunan konveks, dan untuk suatu PL, solusi optimal hanya didapat pada titik-titik ekstrim. Sehingga, walaupun secara teoritis solusi layak tak terhingga banyaknya, akan tetapi untuk menentukan solusi optimal hanya perlu mengobservasi pada titik-titik ekstrim saja. Sedangkan solusi layak suatu IP murni merupakan himpunan titik-titik yang merupakan bilangan bulat. Akibatnya, penentuan solusi optimal suatu IP tidak mudah ditentukan, karena mungkin saja banyak solusi optimal lokal yang didapat.

Secara umum metode untuk menyelesaikan masalah optimisasi kombinatorial terbagi menjadi dua yaitu metode *Exact* dan metode *heuristic*. Metode *exact* mampu, setidaknya secara teori, untuk memberikan solusi optimal, yaitu solusi layak yang mengoptimalkan (meminimalkan atau memaksimalkan) nilai fungsi tujuan, sedangkan metode *heuristic* memberikan solusi yang layak tanpa memberikan adanya jaminan optimalitas, akan tetapi secara waktu jauh kecepatannya melebihi metode *exact*.

3.2 Metode *Exact*

Metode ini menjamin mendapatkan solusi optimal, akan tetapi waktu yang diperlukan untuk menjalankan algoritma (*running time*) mungkin sangat besar karena kompleksitas yang tinggi. Metode ini sangat baik untuk menyelesaikan masalah-masalah dengan ukuran (*size*) kecil. Contoh-contoh metode ini adalah: *branch and bound*, *branch and cut*, *cutting plane*, dan lain lain.

3.2.1 Metode *Branch and bound*

Pada metode *Branch and Bound*, ide dasarnya adalah untuk membagi atau mempartisi himpunan solusi yang layak menjadi himpunan bagian yang lebih kecil secara berurutan. Hal ini dilakukan agar batas menjadi lebih ketat. Untuk masalah minimasi, batas atas adalah nilai terendah dari setiap titik bilangan bulat layak yang ditemui. Metode ini dimulai dengan merelaksasi masalah awal dan mempertimbangkan program linear yang terkait.

Karena dalam masalah optimasi kombinatorial kita mencari solusi yang variabelnya merupakan bilangan bulat, maka jika solusi untuk masalah relaksasi adalah bilangan bulat,

maka solusi tersebut adalah optimal. Jika tidak, solusi itu memberikan batasan dari masalah yang akan diselesaikan.

Metode *Branch and Bound* (BB) digunakan secara luas untuk menyelesaikan masalah-masalah opmimisasi kombinatorial. Karena optimisasi kombinatorial mungkin saja mempunyai banyak solusi yang layak, maka diperlukan waktu proses yang cukup lama untuk mengenumerasi semua solusi layak tersebut. Pada metode BB, ide dasarnya adalah membagi atau memecah solusi yang layak yang ditetapkan ke dalam himpunan bagian yang lebih kecil secara berturut-turut untuk mengembangkan batasan. Untuk masalah minimalisasi, batas atas adalah nilai terendah dari setiap titik integer layak yang didapat.

Metode ini dimulai dengan merelaksasikan masalah awal (original) sehingga menjadi bentuk Program Linear yang terkait (Program Linear Relaksasi dari masalah awal). Jika solusi dari Program Linear Relaksasi tersebut variabelnya adalah integer, maka solusi tersebut merupakan solusi optimal, jika tidak, maka solusi itu merupakan *bound* (batasan) dari masalah. Dengan melakukan *branching* (percabangan) pada variabel yang berupa pecahan pada solusi tersebut, dihasilkan dua sub-masalah.

Misalkan masalah awal yang ingin diselesaikan adalah sebagai berikut:

Obj : Min $\mathbf{c}^T \mathbf{x}$

Kendala: $\mathbf{Ax = b}$

$\mathbf{x \geq 0}$

$x_j \in Z, j = 1, 2, \ldots, n$

Misal x_j adalah variabel berupa pecahan dari solusi yang telah didapat tersebut. Proses *branching* yang dilakukan untuk mendapatkan dua submasalah adalah sebagai berikut :

Submasalah 1	Submasalah 2
Obj : Min $\mathbf{c}^T \mathbf{x}$	Obj : Min $\mathbf{c}^T \mathbf{x}$
Kendala: $\mathbf{Ax = b}$	Kendala: $\mathbf{Ax = b}$
$x_j \leq \lfloor x_j \rfloor$	$x_j \geq \lceil x_j \rceil$
$\mathbf{x \geq 0}$	$\mathbf{x \geq 0}$
$x_j \in Z$, j =1, 2, ..., n	$x_j \in Z$, j =1, 2, ..., n

Pilih salah satu submasalah. Setelah itu submasalah yang telah dipilih tadi diselesaikan. Jika solusi yang didapat variabelnya merupakan pecahan, maka proses *branching* dilakukan kembali dengan cara yang sama seperti di atas. Semua sub-sub masalah harus diselesaikan. Proses investigasi terhadap sub-sub masalah dan membandingkan solusi di sub-sub masalah dihentikan jika proses menemui kriteria penghentian (*stopping criteria*). Adapun kriteria penghentian dari metode *branch and bound* adalah:

(i) Submasalah tidak layak (tidak ada solusi layak).
(ii) Semua variabel pada solusi merupakan bilangan bulat
(iii) Submasalah tidak diinvestigasi karena solusi tidak akan lebih baik dari yang telah didapat pada submasalah sebelumnya.

Secara umum algoritma *branch and bound* dapat dinyatakan sebagai berikut :

Langkah 1:
Buat bentuk Program Linear Relaksasi dari masalah awal. Jika solusi tersebut variabelnya integer, STOP. Solusi optimal didapat.

Langkah 2:
Buat dua submasalah dengan cara melakukan *branching* pada variabel yang berupa pecahan.

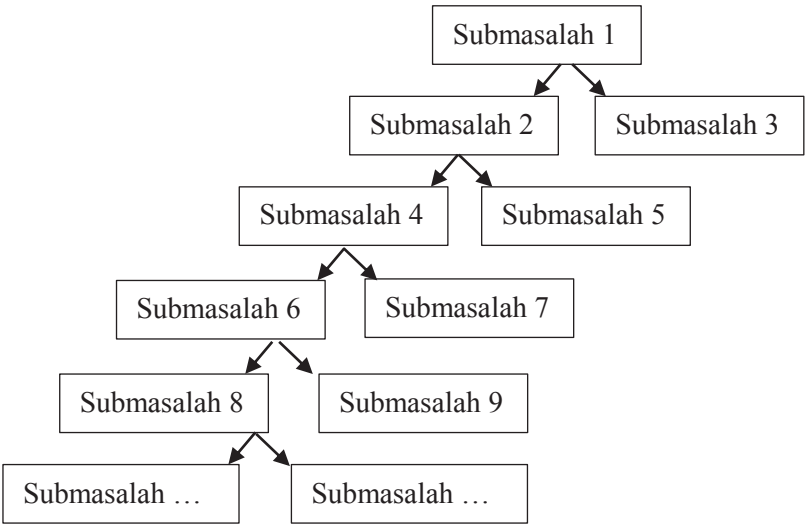
Langkah 3:
Pilih satu submasalah dari Langkah 2. Selesaikan. Jika solusi masih memuat variable berupa pecahan, ulang Langkah 2.

Langkah 4:
Branching tidak lagi dilakukan pada suatu submasalah jika submasalah tersebut telah diinvestigasi atau salah satu dari tiga kriteria penghentian di atas terpenuhi. Ulangi proses ini sampai tidak ada lagi submasalah yang belum diinvestigasi.

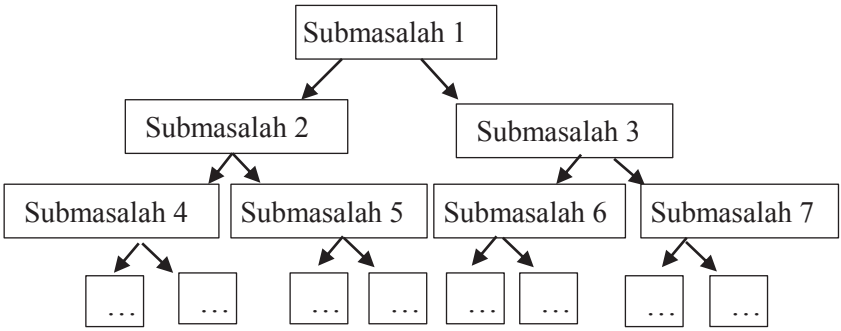
Dalam banyak terapan penggunaan metode *branch and bound*, penentuan batas atas (*upper bound*) maupun batas bawah (*lower bound*) didapat dengan menggunakan *heuristic*. Suksesnya penggunaan BB sangat bergantung kepada strategi *branching*, strategi pencarian (*searching strategy*), dan kualitas dari batas atas atau batas bawah.

Secara umum strategi pencarian dalam metode *branch and bound* dapat dilakukan dengan teknik *depth-first search* (DFS) atau *Breadth First Search* (BFS). Pada teknik DFF proses pencarian dimulai dari titik asal (parent) ke titik percabangan terdalam yang belum diinvestigasi. Ilustrasi percabangan pada teknik DFS dapat dilihat pada Gambar 31.

Untuk teknik BFS (*breadth-first search*), strategi pencarian dilakukan pada level yang sama, baru setelah semua submasalah pada level yang sama selesai, *branching* dan pencarian dilakukan untuk submasalah pada level berikutnya.



Gambar 31. Proses pencarian dengan menggunakan teknik DFS



Gambar 32. Proses pencarian dengan menggunakan teknik BFS

Kelemahan dari metode *branch and bound* adalah waktu komputasi dan penggunaan memory karena metode *branch and bound* mungkin saja memerlukan proses yang lama untuk menginvestigasi semua titik.

Berikut ini diberikan ilustrasi penyelesaian masalah dengan menggunakan metode *branch and bound* yang diambil dari buku *Operations Research Applications and Algorithms* dari W. L. Winston dengan sedikit melakukan modifikasi:

Telfa Corporation memproduksi meja dan kursi. Satu unit meja memerlukan 1 jam tenaga kerja dan kayu/papan seluas 3 meter persegi, dan satu unit kursi memerlukan 1 jam tenaga kerja dan kayu/papan seluas $\frac{5}{3}$ meter persegi. Saat ini, 6 jam kerja dan 15 meter persegi kayu tersedia. Setiap unit meja yang dijual menghasilkan $8 keuntungan, dan setiap kursi yang dijual menghasilkan $5 keuntungan. Formulasikan masalah ini sebagai masalah IP dan selesaikanlah dengan metode *branch and bound* untuk memaksimalkan keuntungan Telfa.

Penyelesaian:

Misalkan: x_1 = banyaknya (unit) meja yang diproduksi.
x_2 = banyaknya (unit) kursi yang diproduksi.

Karena yang diinginkan oleh Telfa adalah mendapatkan keuntungan dari penjualan produk, maka fungsi tujuannya adalah memaksimalkan keuntungan, sehingga bentuk formulasi dari masalah ini adalah sebagai berikut:

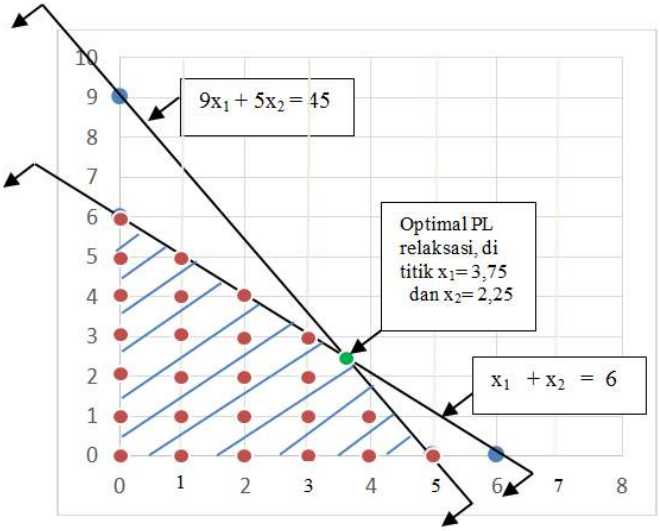
Obj. Max Z = $8x_1 + 5x_2$ (dalam $)
Kendala: $x_1 + x_2 \leq 6$ (kendala tenaga kerja)
$3x_1 + \frac{5}{3} x_2 \leq 15$ (kendala bahan baku papan/kayu)
$x_1 \geq 0, x_2 \geq 0$; x_1, x_2 bilangan bulat.

Perhatikanlah bahwa persyaratan variabel x_1, x_2 merupakan bilangan bulat. Untuk memudahkan perhitungan, kendala kedua dapat diubah menjadi $9x_1 + 5 x_2 \leq 45$, sehingga masalah tersebut dapat diformulasikan sebagai berikut:

IP awal : Obj. Max Z = $8x_1 + 5x_2$
Kendala: $x_1 + x_2 \leq 6$
$9x_1 + 5 x_2 \leq 45$
$x_1 \geq 0, x_2 \geq 0$; $x_1, x_2 \in \mathbb{Z}$

Langkah 1: Bentuk Program Linear relaksasi dari masalah IP awal tersebut adalah dengan me 'relaks' variabel x_1, x_2 menjadi bilangan real, sehingga terbentuk Program Linear relaksasi berikut:

PL relaksasi : Obj. Max Z = $8x_1 + 5x_2$
Kendala: $x_1 + x_2 \leq 6$
$9x_1 + 5 x_2 \leq 45$
$x_1 \geq 0, x_2 \geq 0$; $x_1, x_2 \in R$



Gambar 33. Grafik masalah Telfa Corporation dan solusi optimal Program Linear Relaksasi.

Tanda panah (↙) pada ujung-ujung garis menunjukkan daerah layak dari pertidaksamaan garis. Daerah yang diarsir dengan garis berwarna hijau (⁄) merupakan daerah layak dari PL relaksasi, dan titik titik yang berwarna ● merupakan daerah layak dari IP awal.

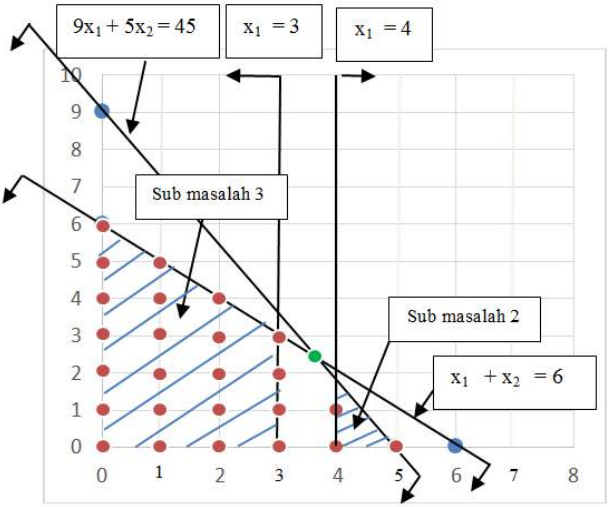
Solusi optimal dari PL relaksasi ini ada di titik $x_1 = 3,75$ dan $x_2 = 2,25$ dengan nilai Z = $\frac{165}{4}$, hal ini berarti nilai $\frac{165}{4}$ merupakan nilai *upper bound* dari masalah Telfa, dengan kata lain, nilai optimal keuntungan Telfa tidak akan melebihi $\frac{165}{4}$. Karena variabel pada solusi masih merupakan pecahan, maka kita ke Langkah 2 dengan melakukan *branching* pada salah satu variabel.

Misalkan kita lakukan branching pada variabel x_1. Maka, kita akan membadi daerah layak menjadi dua daerah dengan menambahkan $x_1 \geq 4$ masalah IP awal (menjadi Sub masalah 2) dan menambahkan $x_1 \leq 3$ pada masalah IP awal (menjadi Sub masalah 3).

Sub masalah 2 : Obj. Max $Z = 8x_1 + 5x_2$
Kendala: $x_1 + x_2 \leq 6$
$9x_1 + 5x_2 \leq 45$
$x_1 \geq 4$
$x_1 \geq 0, x_2 \geq 0 ; x_1, x_2 \in R$

Sub masalah 3 : Obj. Max $Z = 8x_1 + 5x_2$
Kendala: $x_1 + x_2 \leq 6$
$9x_1 + 5x_2 \leq 45$
$x_1 \leq 3$
$x_1 \geq 0, x_2 \geq 0 ; x_1, x_2 \in R$

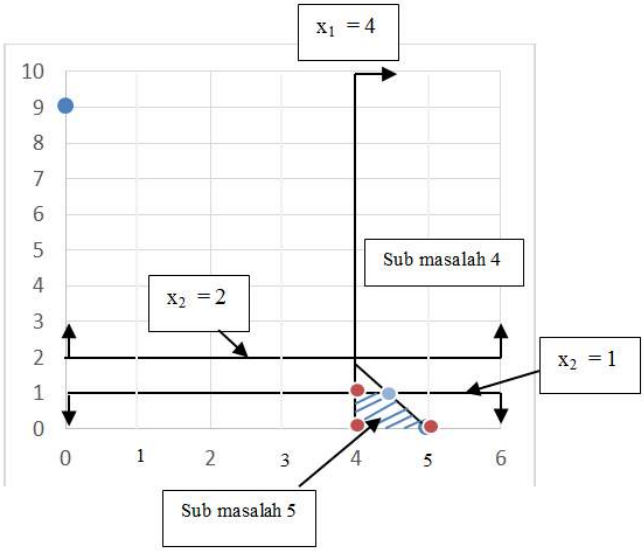
Perhatikanlah bahwa nilai $x_1 = 3{,}75$ tidak termuat sebagai titik di daerah layak, baik pada Sub masalah 2 maupun Sub masalah 3. Dengan kata lain, dengan melakukan *branching*, maka solusi optimal yang didapat sejauh ini menjadi tidak layak lagi.



Gambar 34. Grafik setelah dilakukan proses *branching* pertama (pembentukan Sub masalah 2 dan Sub masalah 3)

Misalkan kita memilih untuk menyelesaikan Sub masalah 2. Solusi optimal dari Sub masalah 2 adalah di titik $x_1 = 4$ dan $x_2 = \frac{9}{5}$ dengan nilai Z = 41.

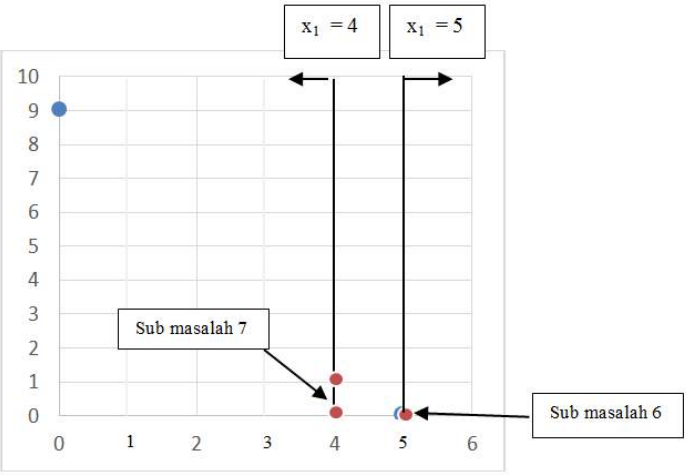
Ternyata Sub masalah 2 juga masih menghasilkan solusi yang variabelnya berupa pecahan, yaitu x_2. Dengan menggunakan teknik *depth first search*, maka kita lakukan lagi *branching* pada variavel x_2 dengan menambahkan kendala $x_2 \geq 2$ pada Sub masalah 2 (menghasilkan Sub masalah 4) dan menambahkan kendala $x_2 \leq 1$ pada Sub masalah 2 (menghasilkan Sub masalah 5).



Gambar 35. Grafik Sub masalah 4 dan Sub masalah 5.

Perhatikanlah bahwa pasa Sub masalah 4, tidak ada daerah layak (*infeasible*) karena tidak ada daerah yang merupakan perpotongan Submasalah 2 dengan kendala $x_2 \geq 2$, sehingga investigasi selanjutnya dilakukan pada Sub masalah 5. Solusi optimal Sub masalah 5 ada pada titik $x_1 = \frac{40}{9}$ dan $x_2 = 1$ dengan nilai $Z = \frac{365}{9}$.

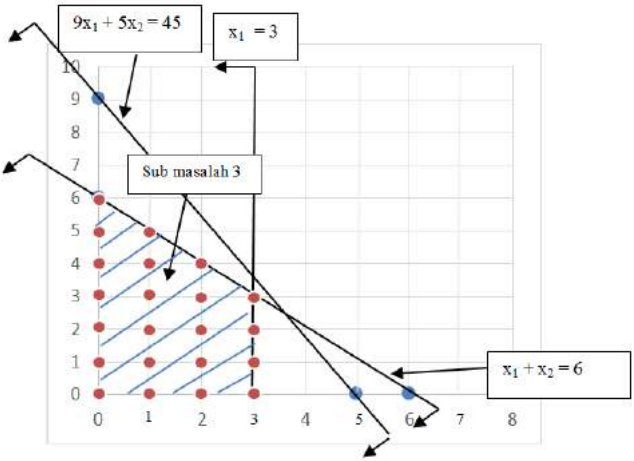
Sub masalah 5 masih menghasilkan solusi yang variabelnya merupakan pecahan, sehingga proses *branching* masih dilakukan, saat ini proses *branching* dilakukan terhadap variabel x_1 yang bernilai $\frac{40}{9}$. Penambahkan kendala $x_1 \geq 5$ pada Sub masalah 5 akan menghasilkan Sub masalah 6, dan penambahan kendala $x_1 \leq 4$ pada Sub masalah 5 akan menghasilkan Sub masalah 7.



Gambar 36. Grafik Sub masalah 6 dan Sub masalah 7

Solusi optimal Sub masalah 6 ada di titik x_1=5 dan x_2=0 dengan nilai Z = 40, dan solusi optimal Sub masalah 7 ada di titik x_1=4 dan x_2= 1 dengan nilai Z = 37. Perhatikanlah bahwa baik solusi optimal pada Sub masalah 6 maupun Sub masalah 7 variabelnya sudah merupakan bilangan bulat, sehingga tidak diperlukan lagi adanya *branching*.

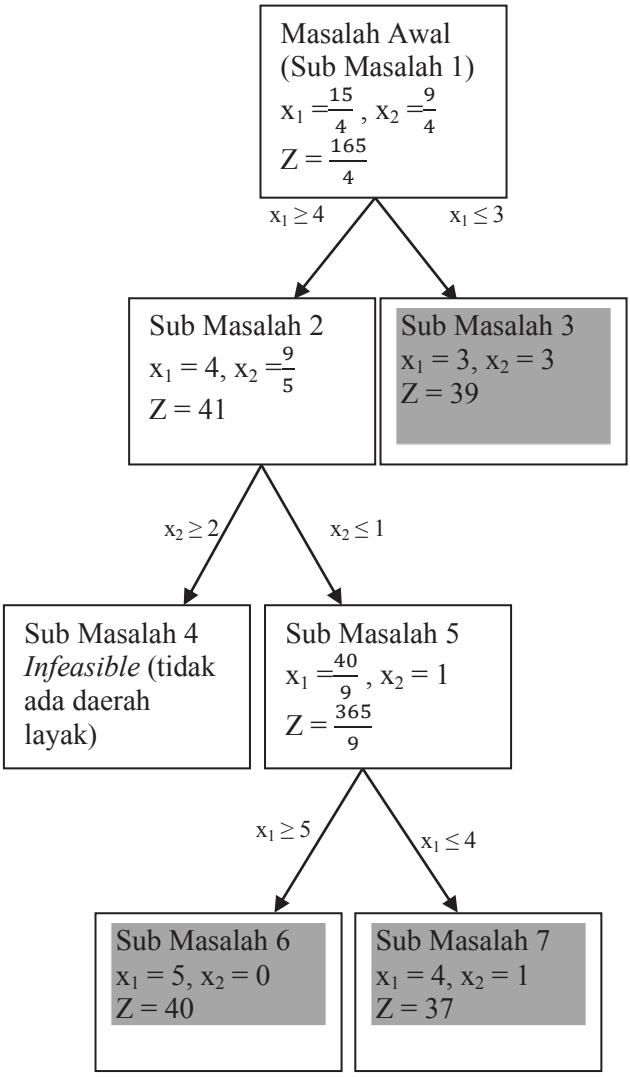
Selanjutnya, kita lihat Sub masalah mana yang belum diinvestigasi. Sub masalah 3 yang belum diinvestigasi.



Gambar 37. Grafik Sub masalah 3.

Solusi optimal dari Sub masalah 3 ada di titik x_1= 3 dan x_2= 3 dengan nilai Z = 39.

Proses ini dapat digambarkan dalam diagram sebagai berikut:



Gambar 38. Ilustrasi proses branching dan penentuan solusi metode *branch and bound*.

Karena semua Sub masalah telah diinvestigasi, maka proses selesai, dan untuk menentukan solusi optimal adalah dengan membandingkan solusi optimal pada Sub masalah Sub masalah yang mendapatkan solusi integer.

Terdapat tiga kandidat solusi optimal (solusi yang variabelnya integer), dan karena masalahnya adalah memaksimalkan, maka solusi optimal didapat pada Sub masalah 6 dengan solusi $x_1 = 5$ dan $x_2 = 0$ dengan nilai $Z = 40$.

Ini berarti, Telfa akan mendapat keuntungan sebesar $40 dengan menjual sebanyak 5 meja dan tidak menjual kursi.

3.2.2 Metode *Cutting Plane*.

Metode *Cutting plane* merupakan salah satu metode untuk menyelesaikan masalah optimisasi kombinatorial dengan cara menambahkan kendala kepada masalah awal. Kendala yang baru ditambahkan ini disebut dengan *cut*. Metode ini dikenalkan oleh Ralph Gomory pada tahun 1950 untuk menyelesaikan masalah *integer programming* dan *mixed integer programming*. Pada awalnya, metode ini dianggap kurang efisien karena banyaknya *cut* yang diperlukan untuk menghasilkan solusi optimal. Akan tetapi, pada tahun 1990 Gérard Cornuéjols menunjukkan bahwa metode *cutting plane* akan sangat efisien jika dikombinasikan dengan metode *branch and bound* (metode yang dihasilkan disebut dengan *branch and cut*). Metode *cutting plane* ini efisien jika *cut* yang digenerate dihasilkan dari tabel metode simpleks. Berikut ini diberikan langkah-langkah metode *cutting plane*.

Langkah 1:
Relaksasikan masalah *Integer Programming* menjadi Program Linear Relaksasi.

Langkah 2:
Selesaikan dengan menggunakan Metode Simpleks. Jika solusi optimal variable yang dipersyaratkan integer telah memenuhi syarat (bernilai integer) maka STOP. Jika tidak, ke Langkah 3.

Langkah 3:
Tentukanlah kendala di tabel solusi optimal tabel simpleks yang ruas kanannya mempunyai nilai pecahan yang mendekati ½. Kendala ini akan digunakan untuk membangkitkan /generate *cut*).

Langkah 4:
Tulis kendala yang ditentukan pada Langkah 3 dalam bentuk $\lfloor x \rfloor + f$, dengan $0 \leq f < 1$.

Langkah 5:
Tulis ulang kendala dalam bentuk:
Ruas kiri berisi semua suku yang koefisiennya berupa integer/ bilangan bulat = Ruas kanan semua suku yang koefisiennya berupa pecahan. *Cut* yang terbentuk adalah :
semua suku dengan koefisiennya pecahan ≤ 0.

Langkah 6:
Tambahkan '*cut*' yang baru terbentuk ke dalam masalah awal. Selesaikan dengan metode dual simpleks. Jika solusi yang didapat variabelnya merupakan bilangan bulat, STOP, jika tidak kembali ke Langkah 2.

Untuk memberikan ilustrasi bagaimana cara kerja metode *cutting plane*, perhatikan masalah yang dihadari oleh Telfa seperti pada contoh sebelumnya.

IP awal : Obj. Max $Z = 8x_1 + 5x_2$

Kendala: $x_1 + x_2 \leq 6$

$9x_1 + 5x_2 \leq 45$

$x_1 \geq 0, x_2 \geq 0 ; x_1, x_2 \in \mathbb{Z}$

Langkah 1: Relaksasi IP menjadi Program Linear relaksasi:

PL relaksasi : Obj. Max $Z = 8x_1 + 5x_2$

Kendala: $x_1 + x_2 \leq 6$

$9x_1 + 5x_2 \leq 45$

$x_1 \geq 0, x_2 \geq 0 ; x_1, x_2 \in R$

Langkah 2: Dengan menyelesaikan masalah tersebut dengan metode simpleks didapat hasil dalam bentuk tabel akhir metode simpleks sebagai berikut:

Tabel 3. Tabel akhir simpleks

Variabel dasar	x_1	x_2	s_1	s_2	Ruas kanan
x_2	0	1	2,25	-0,25	2,25
x_1	1	0	-1,25	0,25	3,75
Z	0	0	1,25	0,75	41,25

Dari tabel simpleks tersebut didapat solusi dengan nilai Z = 41,25 dan $x_1 = 3,75$ serta $x_2 = 2,25$.

Solusi ini masih memuat variabel yang bernilai pecahan, sehingga kita teruskan ke Langkah 3.

Langkah 3: Kita akan tentukan kendala yang nilai ruas kanannya mempunyai pecahan yang mendekati ½ . Karena kedua kendala mempunyai nilai yang sama dekatnya dengan ½, maka misalkan kita ambil kendala kedua yaitu:

$x_1 - 1{,}25\, s_1 + 0{,}25\, s_2 = 3{,}75$

Langkah 4: Tulis dalam bentuk $\lfloor x \rfloor + f$ sehingga didapat :

$x_1 + \lfloor -1{,}25 \rfloor\, s_1 + f\, s_1 + \lfloor 0{,}25 \rfloor\, s_2 + f\, s_2 = \lfloor 3.75 \rfloor + f$

$x_1 - 2\, s_1 + 0.75\, s_1 + 0\, s_2 + 0.25\, s_2 = 3 + 0.75$

Langkah 5: Tulis ulang kendala, didapat :

$x_1 - 2\, s_1 - 3 = 0.75 - 0.75\, s_1 - 0.25\, s_2$

Digenerate *cut* : $0.75 - 0.75\, s_1 - 0.25\, s_2 \leq 0$

Dengan mengganti $s_1 = 6 - x_1 - x_2$ dan $s_2 = 45 - 9x_1 - 5x_2$, maka *cut* $0.75 - 0.75\, s_1 - 0.25\, s_2 \leq 0$ menjadi :

$0.75 - 0.75\,(6 - x_1 - x_2) - 0.25\,(45 - 9x_1 - 5x_2) \leq 0.$

Didapat : $3x_1 + 2x_2 \leq 15$

Cut $-0.75\, s_1 - 0.25\, s_2 \leq -0.75$ ini kemudian ditambahkan ke masalah awal sehingga didapat tabel sebagai berikut:

Tabel 4. Tabel simpleks setelah penambahan *cut*.

Variabel dasar	x_1	x_2	s_1	s_2	s_3	Ruas kanan
x_2	0	1	2,25	-0,25	0	2,25
x_1	1	0	-1,25	0,25	0	3,75
S_3	0	0	-0,75	-0,25	1	-0,75
Z	0	0	1,25	0,75	0	41,25

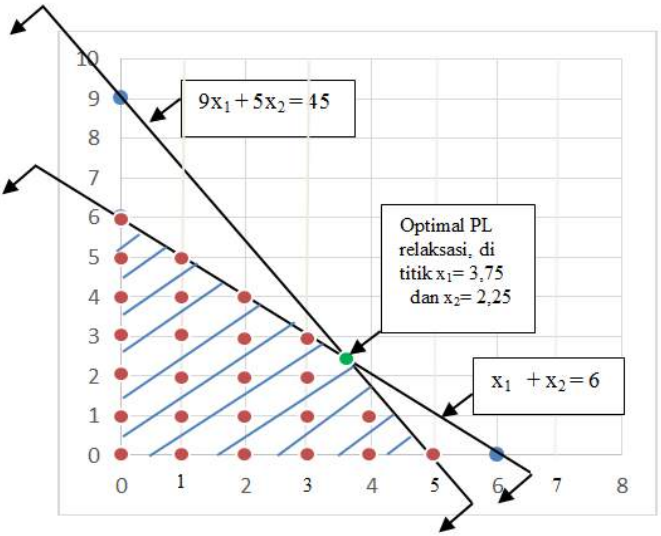
Karena di ruas kanan ada nilai yang negatif, maka dengan menggunakan metode dual simpleks didapat hasil sebagai berikut:

Tabel 5. Tabel akhir simpleks setelah penambahan *cut*.

Variabel dasar	x_1	x_2	s_1	s_2	s_3	Ruas kanan
x_2	0	1	0	-1	3	0
x_1	1	0	0	0,67	-1,67	5
s_1	0	0	1	0,33	-1,33	1
Z	0	0	0	0,33	1,67	40

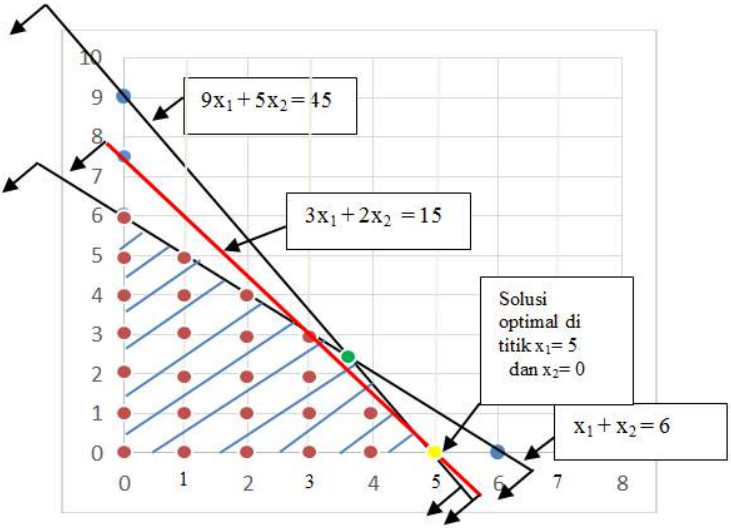
Dari tabel di atas dapat dilihat bahwa solusi telah menghasilkan variabel bernilai integer sehingga solusi optimal didapat dengan nilai x_1=5, x_2=0 dan Z = 40.

Masalah ini dapat juga digambarkan dengan menggunakan grafik sebagai berikut:



Gambar 39. Solusi optimal untuk Program Linear Relaksasi

Dari grafik dapat dilihat bahwa nilai optimal PL relaksasi, di titik x_1= 3,75 dan x_2= 2,25 dengan nilai Z = 41,25. Setelah ditambahkan *cut* $3x_1 + 2x_2 \leq 15$ diperoleh:



Gambar 40. Grafik setelah ditambahkan *cut*.

Perhatikanlah bahwa penambahan *cut* $3x_1+2x_2 \leq 15$ menghasilkan dua solusi dengan variabel integer, yang pertama perpotongan *cut* dengan garis $x_1 + x_2 \leq 6$ menghasilkan solusi di titik (3,3) dengan nilai Z = 37, dan perpotongan *cut* dengan garis $9x_1 + 5x_2 \leq 45$ menghasilkan solusi dititik (5,0) dengan nilai Z = 40. Nilai optimal di Z = 40.

Jika penambahan *cut* tidak menghasilkan solusi integer, maka lakukan penambahan *cut* terus menerus sampai solusi integer didapat. Perhatikanlah bahwa penambahan *cut* akan menyebabkan solusi optimal dari PL relaksasi menjadi tidak layak (*cut* mengeliminir solusi optimal dari PL relaksasi yang merupakan pecahan).

3.2.3 Metode Relaksasi Lagrange

Beberapa masalah optimisasi kombinatorial menjadi sulit diselesaikan jika masalah tersebut ditambah dengan parameter atau kendala baru. Sebagai contoh, MST merupakan masalah yang masuk dalam kelas P, akan tetapi jika ditambahkan kendala batasan *degree* yang harus dipenuhi pada tiap titiknya, maka masalah tersebut menjadi NP-*complete*. Kadang-kadang, beberapa masalah menjadi sulit diselesaikan secara eksplisit karena adanya kendala yang sulit tersebut. Untuk mengatasi hal tersebut, dikenalkan suatu metode yang disebut dengan Relaksasi Lagrange.

Bentuk umum *Integer Programming* dapat ditulis sebagai berikut:

(P) Obj. Minimize $\mathbf{cx}$

Kendala: $\mathbf{Ax} \geq \mathbf{b}$

$\mathbf{Bx} \geq \mathbf{d}$

$x_j \geq 0$, x_j integer, $j \in I$.

dengan **b**, **c**, dan **d** adalah vektor dan **A** dan **B** adalah matriks yang bersesuaian, I adalah indeks yang menunjukkan bahwa variable x_j harus merupakan integer. Pada model ini diberikan dua jenis kendala yaitu $\mathbf{Ax} \geq \mathbf{b}$ dan $\mathbf{Bx} \geq \mathbf{d}$ dengan alasan bahwa kendala jenis kedua yaitu $\mathbf{Bx} \geq \mathbf{d}$ adalah kendala dengan struktur khusus.

Definisikan bentuk Relaksasi Lagrange dari (P) terhadap $\mathbf{Ax} \geq \mathbf{b}$ dan vektor $\boldsymbol{\lambda}$ yang bersesuaian sebagai:

(P_λ): Obj. Minimize $\mathbf{cx} + \boldsymbol{\lambda}(\mathbf{b} - \mathbf{Ax})$

Kendala: $\mathbf{Bx} \geq \mathbf{d}$

$x_j \geq 0$, x_j integer, $j \in I$.

Bentuk P_λ lebih mudah diselesaikan daripada bentuk (P).

Contoh:
Diberikan masalah *Set Cover* berikut:

$$A = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 1 \end{pmatrix} \text{ dengan } c_j = (2, 3, 4,5)$$

Kolom 1 dengan biaya 2 meng *cover* (menutupi) baris 1 dan baris 2, kolom 2 dengan biaya 3 meng *cover* baris 3, kolom 3 dengan biaya 4 meng *cover* baris 1 dan baris 3, kolom 4 dengan biaya 5 meng *cover* baris 2 dan baris 3.

Diinginkan jumlah kolom yang dapat meng *cover*/ menutupi semua baris dengan total biaya minimum.

Penyelesaian:
Masalah tersebut dapat dimodelkan sebagai berikut:
Misalkan:

$$x_j = \begin{cases} 1, \text{jika kolom j merupakan bagian dari solusi} \\ 0, \text{selain itu} \end{cases}$$

Obj. Min $2x_1 + 3x_2 + 4x_3 + 5x_4$
Kendala:

$x_1 + x_3 \geq 1$
$x_1 + x_4 \geq 1$
$x_2 + x_3 + x_4 \geq 1$

Untuk mengimplementasikan Metode Relaksasi Lagrange ke masalah ini, kita kalikan vektor pengali Lagrange $\lambda \geq 0$ dengan ketiga kendala dan memasukkannya ke fungsi tujuan sehingga didapat:

Obj. Min $2x_1 + 3x_2 + 4x_3 + 5x_4 + \lambda_1(1- x_1 - x_3) + \lambda_2(1- x_1 - x_4)$
$+ \lambda_2(1- x_2 - x_3 - x_4)$

Kendala:

$x_j \in \{0, 1\}, j = 1, 2, 3, 4.$

Model ini dapat juga dibuat sebagai berikut:

Obj. Min $(2 - \lambda_1 - \lambda_2)\, x_1 + (3 - \lambda_2)\, x_2 + (4 - \lambda_1 - \lambda_3)\, x_3 +$
$(5 - \lambda_2 - \lambda_3)\, x_4 + \lambda_1 + \lambda_2 + \lambda_3$

Kendala:

$x_j \in \{0, 1\}, j = 1, 2, 3, 4$

Dengan mengganti

$C_1 = 2 - \lambda_1 - \lambda_2$

$C_2 = 3 - \lambda_2$

$C_3 = 4 - \lambda_1 - \lambda_3$

$C_4 = 5 - \lambda_2 - \lambda_3$

didapat

Obj. Min $C_1 x_1 + C_2 x_2 + C_3 x_3 + C_4 x_4 + \lambda_1 + \lambda_2 + \lambda_3$

Kendala:

$x_j \in \{0, 1\}, j = 1, 2, 3, 4$

$$x_j = \begin{cases} 1, \text{jika } C_j \leq 0 \\ 0, \text{selain itu} \end{cases}$$

Nilai dari $Z = C_1 x_1 + C_2 x_2 + C_3 x_3 + C_4 x_4 + \lambda_1 + \lambda_2 + \lambda_3$ merupakan nilai dari *lower bound* dari solusi yang telah didapat. Masalah yang dihadapi adalah: bagaimana menentukan nilai λ_1, λ_2, λ_3 untuk mendapatkan nilai *lower bound* yang baik? Untuk masalah di atas, misalkan kita ambil nilai λ_1=1,5, nilai λ_2=1,6 dan nilai λ_3=2,2 maka didapat:

$C_1 = 2 - \lambda_1 - \lambda_2 = -1{,}1$
$C_2 = 3 - \lambda_2 = 0{,}8$
$C_3 = 4 - \lambda_1 - \lambda_3 = 0{,}3$
$C_4 = 5 - \lambda_2 - \lambda_3 = 1{,}2$

Karena hanya C_1 yang bernilai negatif, maka solusi adalah $x_1 = 1$, $x_2 = x_3 = x_4 = 0$ dan nilai Z adalah:

$Z = C_1 x_1 + C_2 x_2 + C_3 x_3 + C_4 x_4 + \lambda_1 + \lambda_2 + \lambda_3$
$= -1{,}1 + 0 + 0 + 0 + 1{,}5 + 1{,}6 + 2{,}2 = 4{,}2.$

Nilai 4,2 ini merupakan niai *lower bound* dari solusi optimal.

Perhatikanlah bahwa dengan melakukan inspeksi didapat solusi dengan nilai total 5 dan $x_1 = x_2 = 1$, dan $x_3 = x_4 = 0$.

Branch and Cut

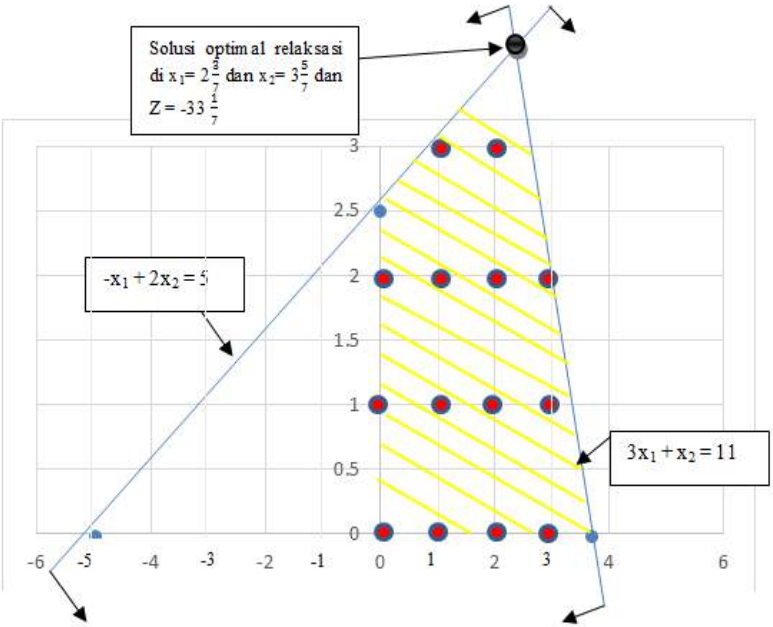
Metode *cutting plane* didasarkan atas ide bahwa jika solusi optimal yang didapat merupakan titik ekstrim, maka solusi optimal didapat. Metode *cutting plane* menyelesaikan masalah dengan memodifikasi masalah awal dengan menambahkan *cut* yang menyebabkan solusi optimal dari program linear relaksasi yang sebelumnya didapat (yang variabelnya ada yang bukan integer) menjadi tidak layak di daerah layak yang baru (mereduksi daerah layak). Penambahan cut ini dilakukan terus menerus sampai solusi optimal didapat.

Sama dengan ide dari metode *cutting plane*, metode *branch and bound* juga mereduksi daerah solusi dan menyebabkan solusi optimal relaksasi yang solusinya masih mempunyai variable non integer menjadi tidak layak lagi di daerah layak yang telah direduksi. Perbedaan mendasar dari

kedua ini adalah pada metode *cutting plane* daerah yang akan diinvestigasi tetap merupakan satu daerah layak, sedangkan pada metode *branch and cut* daerah layak yang telah direduksi terbagi atas beberepa daerah layak karena adanya proses *branching* (percabangan). Akibatnya, pada metode *branch and bound* waktu proses menjadi lebih lama karena semua daerah layak harus diinvestigasi. Oleh karena itu, ide muncul untuk mengkombinasikan metode *branch and bound* dan metode *cutting plane*, dan metode ini dikenal dengan metode *branch and cut*.

Pada metode *branch and cut*, terdapat tiga prosedur utama, yaitu: reformulasi, *heuristic* yang dikembangkan untuk menyelesaikan masalah, dan *cutting plane*. Ketiga prosedur ini ditambahkan ke langkah-langkah yang ada di metode *branch and bound.*

Contoh: diberikan *Integer Programming* sebagai berikut:
Masalah awal:
Obj. Min - $6x_1$ - $5x_2$
Kendala: $3x_1 + x_2 \leq 11$
$-x_1 + 2x_2 \leq 5$
$x_1 \geq 0$, $x_2 \geq 0$, x_1 dan x_2 bilangan bulat.



Gambar 41. Solusi layak Program Linear relaksasi (berupa daerah yang diarsir dengan warna kuning ( ) dengan solusi optimal di titik ($2\frac{3}{7}$, $3\frac{5}{7}$)dan solusi layak bagi integer programming berupa titik-titik yang berwarna

Langkah pertama dari metode *branch* and *cut* adalah sama dengan *branch and bound* yaitu menyelesaikan masalah yang telah direlaksasi. Solusi yang didapat adalah $-33\frac{1}{7}$ dengan $x_1 = 2\frac{3}{7}$ dan $x_2 = 3\frac{5}{7}$.

Karena solusi yang didapat merupakan solusi yang variabelnya berupa pecahan, maka dilakukan proses *branching*. Anggap akan dilakukan *branching* terhadap variable x_1, maka didapat dua Sub masalah sebagai berikut:

Sub Masalah 2

Obj. Min - $6x_1 - 5x_2$

Kendala: $3x_1 + x_2 \leq 11$

$-x_1 + 2x_2 \leq 5$

$x_1 \geq 3$

$x_1 \geq 0, x_2 \geq 0$

x_1 dan x_2 bilangan bulat.

Sub Masalah 3

Obj. Min - $6x_1 - 5x_2$

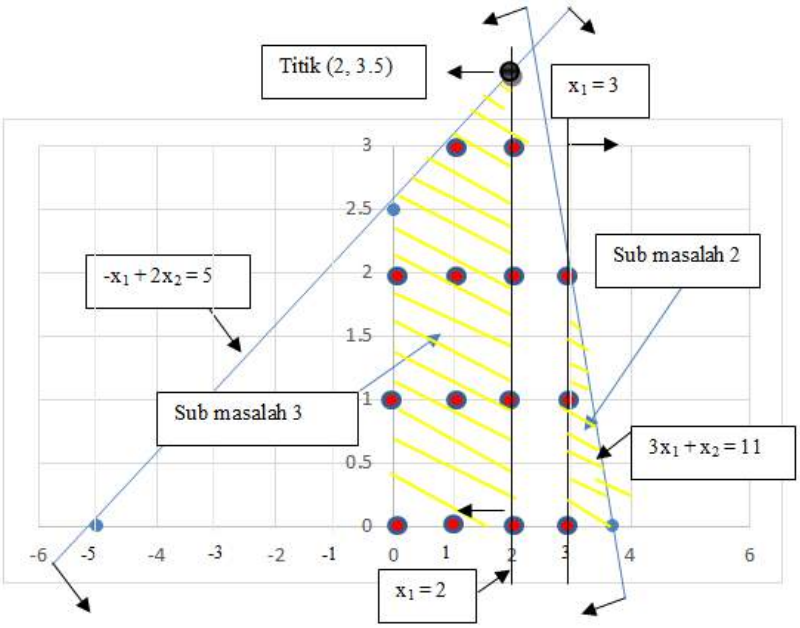
Kendala: $3x_1 + x_2 \leq 11$

$-x_1 + 2x_2 \leq 5$

$x_1 \leq 2$

$x_1 \geq 0, x_2 \geq 0$

x_1 dan x_2 bilangan bulat.



Gambar 42. *Branching* di x_1 untuk mendapatkan Sub masalah 2 dan Sub masalah 3.

Sub masalah 2 menghasilkan solusi di titik (3,2) dengan nilai Z = -28. Karena solusi ini variabelnya sudah merupakan bilangan bulat, maka Sub masalah 2 selesai, tidak perlu dilakukan *branching* dan solusi ini merupakan solusi optimal terbaik saat ini. Sedangkan Sub masalah 3 menghasilkan solusi optimal di titik (2, 3.5) dengan nilai Z= -29,5. Solusi ini variabelnya masih berupa pecahan, sehingga perlu dilakukan proses selanjutnya.

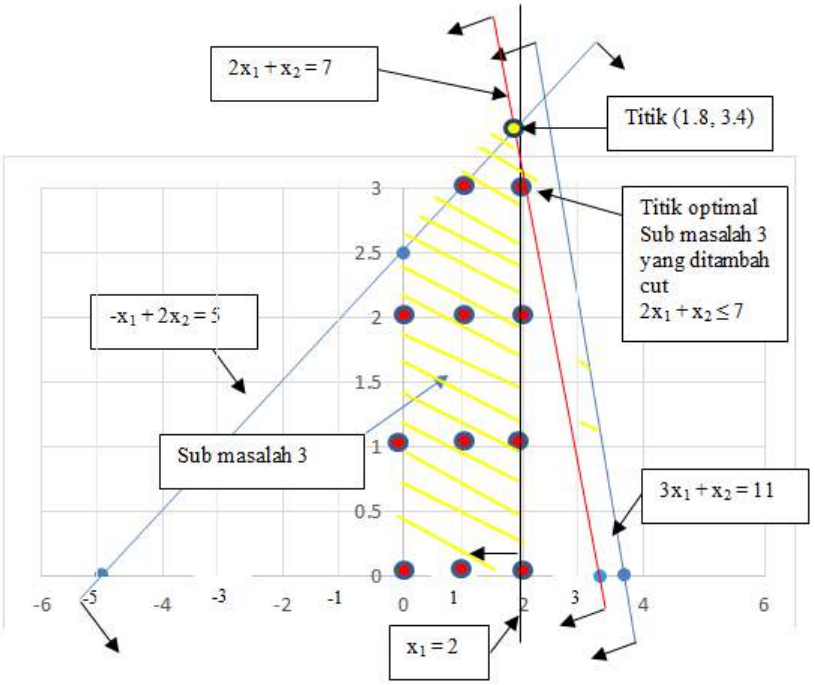
Asumsikan bahwa untuk proses selanjutnya tidak dilakukan *branching* tapi membangkitkan *cut*. Misalkan *cut* yang dibangkitkan adalah $2x_1 + x_2 \leq 7$ yang ditambahkan ke Sub masalah 3. Perhatikanlah bahwa $2x_1 + x_2 \leq 7$ membuat solusi

optimal di titik (2, 3.5) menjadi tidak layak, sehingga memenuhi syarat *cut*, sehingga Sub masalah 3 menjadi:

Obj. Min - $6x_1 - 5x_2$
Kendala: $3x_1 + x_2 \leq 11$
$-x_1 + 2x_2 \leq 5$
$x_1 \leq 2$
$2x_1 + x_2 \leq 7$
$x_1 \geq 0$, $x_2 \geq 0$, x_1 dan x_2 bilangan bulat

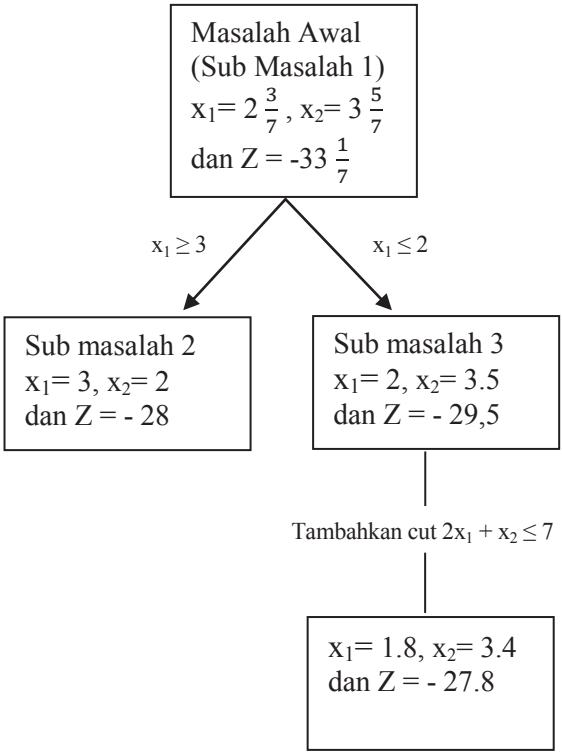
Solusi optimal dari masalah ini didapat di titik (1.8, 3.4) dengan nilai Z = -27.8.

Perhatikanlah bahwa solusi ini tidak lebih baik dari solusi yang telah didapatkan sebelumnya di Sub masalah 2 yang variabelnya sudah bernilai integer dengan Z=-28. Sehingga, dapat diprediksi bahwa pencarian di Sub masalah 3 akan menghasilkan solusi yang tidak lebih baik dari Sub masalah 2.



Gambar 43. Grafik setelah penambahan *cut* $2x_1 + x_2 \leq 7$

Proses *branch and cut* ini dapat digambarkan pada diagram berikut:



Gambar 44. Contoh diagram proses *branch and cut.*

Perhatikanlah bahwa penambahan *cut* pada Sub masalah 3 menghasilkan suatu solusi yang tidak lebih baik dari solusi dengan variable bernilai integer yang telah didapat di Sub masalah 2. Akibatnya, penambahan *cut* ini mereduksi proses sehingga membuat metode menjadi lebih efisien. Sebab, jika dilakukan tanpa penambahan *cut*, maka pada Sub masalah 3 akan dilakukan *branching* lagi dan akan diinvestigasi lagi sub-sub masalah berikutnya.

3.3 *Metode heuristic*

Untuk mengatasi kompleksitas dari permasalahan yang muncul maka para peneliti tertantang untuk mengembangkan suatu metode yang cepat dan efisien yang memberikan solusi dengan kualitas yang baik dan dapat diterima walaupun tidak optimal (solusi ini sering disebut dengan solusi yang hampir optimal atau *nearly optimal*). Foulds pada tahun 1992 menyatakan bahwa Pappas pada awal abad ke 3 telah memberikan ide untuk metode pendekatan ini yang disebut dengan *heuristic* yang tujuannya adalah untuk mengobservasi metode metode menemukan sesuatu, mendeteksi, dan mengembangkan dengan menggunakan logika dan filsafat. Kata *heuristic* berasal dari Bahasa Yunani *heuriskein*, yang berarti menemukan. *Heuristic* atau algoritma pendekatan tidaklah menjamin akan memerikan solusi yang optimal.

Heuristic didesain untuk mendapatkan solusi yang baik, tidak perlu optimal (walaupun kadang-kadang optimal), akan tetapi waktu yang diperlukan sangat cepat jika dibandingkan dengan metode *exact*. Untuk masalah-masalah yang masuk kelas NP-*hard* maupun NP-*complete* dimana algoritma *exact* yang cepat hampir dapat dikatakan tidak ada, maka *heuristic* merupakan alternatif yang sering dikembangkan. Kualitas solusi dari suatu *heuristic* ditentukan oleh seberapa jauh solusi tersebut dari solusi optimal. Kualitas ini (untuk masalah minimisasi) seringkali dihitung dengan menggunakan rumus $\frac{H-Opt}{Opt}$, dimana *H* adalah solusi yang didapat dengan menggunakan *exact* dan *Opt* adalah solusi optimal. Jika solusi optimal tidak diketahui, kualitas tersebut (untuk kasus minimisasi) dapat dihitung dengan menggunakan rumus $\frac{H-LB}{LB}$, LB adalah nilai *lower bound* dari solusi.

Metode *heuristic* memberikan solusi yang subtimal dan tidak menjamin akan memberikan solusi yang optimal (walaupun kadang-kadang solusi yang diberikan juga merupakan solusi optimal). Walaupun *running time* tidak dijamin polinomial, akan tetapi jauh lebih cepat daripada metode *exact*. Oleh karena itu, metode ini cocok digunakan untuk menyelesaikan masalah-masalah dengan ukuran yang besar. Berikut ini akan didiskusikan salah satu metode *heuristic*, yaitu Tabu *Search*.

3.3.1 *Tabu Search*

Tabu Search adalah salah satu metode pencarian/ *searching* yang telah berhasil diterapkan untuk mengatasi beberapa masalah sulit di bidang optimasi kombinatorial antara lain pada penyelesaian masalah jaringan telekomunikasi yang telah diinvestigasi oleh Costamagna dkk. pada tahun 1998 serta Xu dkk pada tahun 1995; *vehicle routing problem* oleh Badeau dkk pada tahun 1995, serta Semet dan Taillard pada tahun 1993, masalah penjadwalan oleh Taillard pada tahun 1994, dan banyak lainnya.

Fred Glover pada tahun 1986 mengusulkan metode *Tabu Search* yang diturunkan dari *neighbourhood searching*. Kata 'Tabu' (awalnya *Taboo*) berasal dari bahasa Tonga di Polinesia, yang digunakan oleh penduduk asli Pulau Tonga. Kata Tabu berarti sesuatu yang harus dihindari karena dianggap suci.

Tabu *Search* dapat diterapkan secara langsung pada pernyataan verbal atau simbolis dari banyak jenis masalah keputusan, tanpa perlu mentransfernya ke dalam formulasi matematika. Namun demikian, sangat berguna untuk memperkenalkan notasi matematika untuk mengekspresikan kelas yang luas dari masalah ini, sebagai dasar untuk

menggambarkan fitur-fitur tertentu dari Tabu *Search*. Konsep dasar matematika dari Tabu *Search* dijelaskan berikut ini.

Notasikan S sebagai himpunan dari solusi. Untuk vektor $u \in S$ definisikan himpunan $N(u)$, sebagai lingkungan (*neighborhood*) dari u, dari solusi-solusi yang 'dekat' u. Banyaknya iterasi dinotasikan dengan k dan S_k adalah himpunan dari solusi pada saat iterasi ke k. Jika f adalah fungsi tujuan, masalah optimissi dapat dinyatakan sebagai berikut:

$$\underset{i \in S}{Minimize}\ f(i).$$

Tabu *Search* mulai dengan cara yang sama seperti *local search*. Proses dilakukan secara berulang menginvestigasi dari satu solusi ke solusi lainnya sampai kriteria penghentian (*stopping criteria*) terpenuhi. Setiap solusi didapat dari u dengan suatu operasi yang disebut dengan *move*.

Tabu *Search* melampui *local search* dengan menggunakan suatu strategi untuk memodifikasi $N(u)$ sewaktu proses *searching* berlangsung yang secara efektif menggantinya dengan *neighborhood* lain $N^*(u)$ dengan menggunakan suatu memori khusus yang berfungsi untuk menentukan $N^*(u)$ pada setiap iterasi serta mengatur bagaimana ruang solusi dieksplorasi. Karena struktur dari *neighborhood* tergantung kepada iterasi, maka digunakan notasi $N(u,k)$ daripada $N^*(u)$.

Definisi dari $N(u,k)$ mengimplikasikan bahwa beberapa solusi yang baru saja dihasilkan akan dibuang dari $N(u,k-1)$. Solusi ini dipertimbangkan sebagai *tabu solutions*, yang harus dihindari pada iterasi berikutnya.

Untuk meningkatkan efisiensi proses eksplorasi, seseorang perlu melacak tidak hanya informasi lokal (seperti nilai fungsi tujuan saat ini) tetapi juga beberapa informasi yang terkait dengan proses eksplorasi. Informasi ini akan digunakan untuk memandu perpindahan dari satu solusi ke solusi lainnya. Penggunaan memori secara sistematis ini merupakan fitur penting dari Tabu *Search*. Cara menetapkan atribut tertentu ke solusi tergantung pada penggunaan struktur memori. Dalam Tabu *Search*, perbedaan penting muncul dengan membedakan antara memori jangka pendek dan memori jangka panjang. Landasan utama Tabu *Search* terletak pada kedua jenis memori tersebut. Setiap jenis memori memiliki strategi khusus sendiri.

Memori Jangka Pendek (*Short Term Memory*)

Memori jangka pendek melacak atribut solusi yang telah berubah selama masa lalu, dan memori ini disebut sebagai memori berbasis kebaruan oleh Glover dan Laguna pada tahun 1997. Memori berbasis kebaruan dieksploitasi dengan menetapkan penunjukan tabu aktif ke atribut terpilih yang muncul dalam solusi yang baru saja dikunjungi, dan kemudian solusi yang mengandung elemen tabu aktif menjadi tabu. Namun, solusi ini hanya menahan status tabu atau atribut tabu untuk sementara, tidak selamanya. Tabu *tenure* didefinisikan sebagai durasi atribut tetap tabu aktif (biasanya diukur dengan jumlah iterasi). Tabu *tenure* dapat bervariasi untuk berbagai jenis atau kombinasi atribut, juga dapat bervariasi pada waktu atau tahap pencarian yang berbeda.

Salah satu istilah yang menyertai *short term memory* adalah *aspiration criteria* (kriteria aspirasi) atau kondisi aspirasi. Status tabu suatu solusi tidak bersifat mutlak, tetapi

dapat diganti jika memenuhi syarat-syarat tertentu, dan hal ini dinyatakan dalam bentuk syarat aspirasi. Akibatnya, kondisi aspirasi ini memberikan peluang atau daya tarik bahwa suatu solusi mungkin dianggap dapat diterima meskipun diklasifikasikan sebagai tabu.

Dalam beberapa aplikasi, komponen *short term memory* cukup untuk menghasilkan solusi berkualitas baik. Namun, secara umum, Tabu *Search* menjadi lebih kuat dengan memasukkan *long term memory* (memori jangka panjang) dan strategi yang terkait.

***Long term Memory* (Memori Jangka Panjang).**

Satu istilah dasar pada *long-term memory* adalah *frequency-based memory* (memori berbasis frekuensi). *Frequency based memory* bekerja dengan mengenalkan penalti/hukuman dan dorongan atau penghargaan yang ditentukan oleh rentang waktu relatif dimana atribut dari solusi yang telah didapat dapat berubah dengan mengubah daerah pencarian. Seberapa sering atribut suatu solusi berubah dilacak oleh *transition frequencies* (frekuensi transisi), sementara *residence frequencies* (frekuensi residen) melacak durasi relatif atribut muncul dalam solusi yang dihasilkan.

***Intensification Strategy* (Strategi Intensifikasi).**

Strategi intensifikasi adalah komponen penting lainnya dalam memori jangka panjang. Strategi intensifikasi didasarkan pada modifikasi aturan pilihan untuk mendorong kombinasi gerakan dan fitur solusi yang secara historis dianggap baik. Strategi intensifikasi juga dapat menginisiasi kembalinya proses pencarian solusi ke daerah-daerah yang menarik untuk ditelusuri lebih mendalam.

***Diversification Strategy* (Strategi Diversifikasi).**

Strategi diversifikasi, seperti namanya, dirancang untuk mendorong pencarian ke wilayah baru. Strategi ini sering didasarkan pada modifikasi aturan pilihan untuk membawa atribut ke dalam solusi yang jarang digunakan, atau sebagai alternatif, mungkin memperkenalkan atribut tersebut dengan memulai kembali sebagian atau seluruhnya proses solusi.

Esensi Tabu *Search* dapat dinyatakan sebagai berikut:
Catatan: i* merupakan solusi terbaik yang tersedia.

Prosedur Tabu *Search*

Langkah 1:
Pilih satu solusi awal $i \in S$. Set $i^* = i$, dan $k=0$.

Langkah 2:
Set $k=k+1$ dan bangkitkan/generate satu himpunan bagian V* dari solusi-solusi di $N(i,k)$ sedemikian sehingga kondisi tabu dilanggar atau paling sedikit satu kondisi aspirasi terpenuhi.

Langkah 3: Pilih j yang terbaik, $j = i \oplus m \in V^*$ terhadap f, dan set $i=j$.

Langkah 4: Jika $f(i) < f(i^*)$ maka set $i^* = i$.

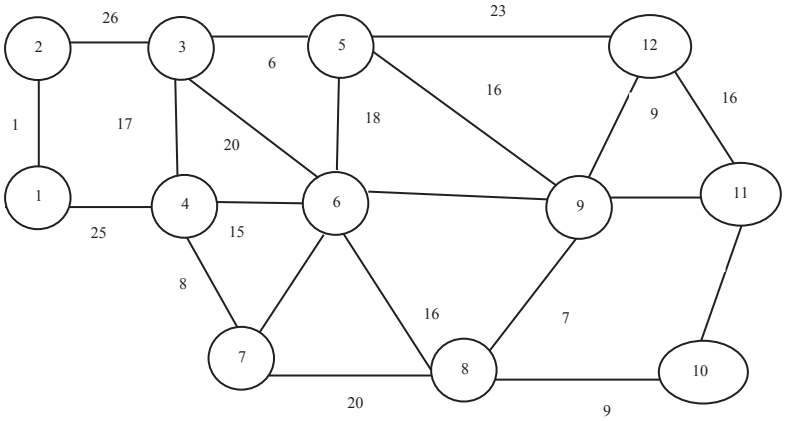
Langkah 5:
Perbarui kondisi tabu dan kondisi aspirasi

Langkah 6:
Jika kriteria penghentian terpenuhi, stop. Jika tidak, Kembali ke Langkah 2.

Berikut ini diberikan satu contoh untuk mengilustrasikan bagaimana cara kerja Tabu *Search* yang diambil dari buku *Tabu Search* yang ditulis oleh F. Glover dan M. Laguna pada tahun 1997.

Contoh:
Diberikan suatu graf berbobot seperti pada Gambar 45. Akan ditentukan minimum *k-tree* dari graf tersebut.

Masalah minimum *k-tree* adalah menentukan suatu *tree* dari suatu graf berbobot yang terdiri dari *k* garis sedemikian sehingga jumlah bobot total dari *k* garis tersebut minimum.



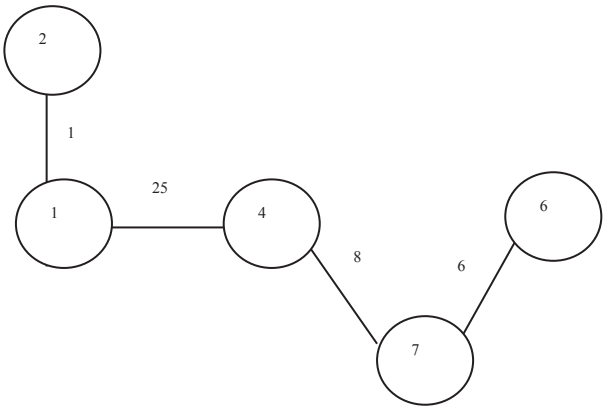
Gambar 45. Contoh graf untuk masalah minimum *k-tree*.

Konstruksi dimulai dengan memilih garis e_{ij} yang memiliki bobot terkecil di graf. Sisa k -1 garis akan dipilih berturut-turut sedemikian sehingga penambahan garis menghasilkan bobot terkecil yang mungkin, dimana garis yang dipertimbangkan tepat bertemu pada satu titik dari titik-titik ujung dari garis yang telah dipilih sebelumnya. Untuk contoh

di atas, anggap kita ingin menentukan minimum *k-tree* dengan k = 4. Tabel berikut menunjukkan prosedur pencarian yang dilakukan.

Tabel 6. Proses pencarian.

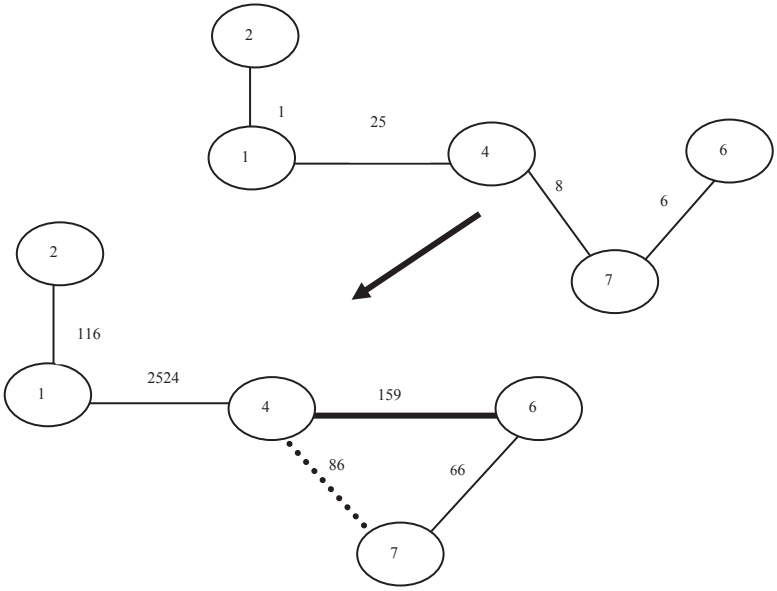
Langkah	Kandidat	Yang dipilih	Total bobot
1	(1,2)	(1,2)	1
2	(1,4), (2,3)	(1,4)	26
3	(2,3),(3,4),(4,6),(4,7)	(4,7)	34
4	(2,3), (3,4),(4,6),(6,7),(7,8)	(6,7)	40



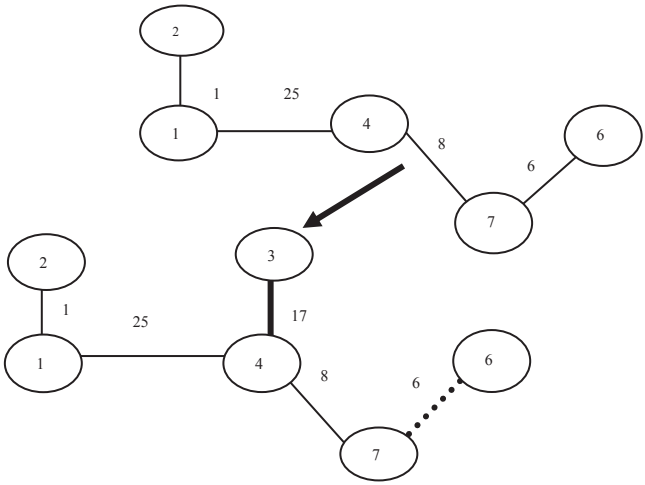
Gambar 46. Solusi awal konstruksi 4-*tree*

Mekanisme operasi ***move*** yang digunakan adalah mengganti satu garis yang ada di 4-*tree* yang didapat dengan yang ada di luar 4-*tree* dengan syarat penggantian garis tersebut tetap menghasilkan *tree* yang banyak garisnya 4.

Terdapat dua jenis penggantian garis pada masalah minimum *k-tree* yaitu penggantian **statis** (*static swap*) dan penggantian **dinamis** (*dynamic swap*). Penggantian statis adalah mengganti garis tetapi tetap mempertahankan titik-titik yang ada di *k-tree*, sementara penggantian dinamis menghasilkan *tree* yang memuat titik baru.



Gambar 47. Penggantian Statis



Gambar 48. Penggantian Dinamis

Anggap untuk masalah ini kita mendefinisikan *tabu tenure* untuk garis yang dibuang adalah 2 (dianggap *tabu-active* untuk dua iterasi, dengan kata lain garis yang baru dibuang dari *k-tree* itu tidak dapat dipilih kembali selama 2 iterasi berikutnya), sementara *tabu tenure* untuk garis yang baru ditambahkan adalah 1 (tidak dapat langsung dibuang, tetapi harus berada di *k-tree* minimal untuk 1 kali iterasi). Diasumsikan juga *move* menjadi tabu jika garis yang barus dibuang atau baru ditambahkan berada pada status *tabu active*.

Dengan menggunakan konstruksi *greedy* untuk menentukan solusi awal, jika kita memeriksa semua lingkungan dari penggantian garis yang mungkin pada tiap iterasi, dan selalu memilih yang terbaik, yang bukan tabu, maka kita dapatkan tabel berikut yang merupakan tiga *move* yang pertama.

Tabel 7. Tiga iterasi pertama dari penentuan 4-*tree*.

Iterasi	*Tabu active*		Ditambahkan	Dibuang	Bobot
	1	2			
1			(4,6)	(4,7)	47
2	(4,6)	(4,7)	(6,8)	(6,7)	57
3	(6,8)	(6,7)	(8,9)	(1,2)	63

Bilangan 1 dan 2 di kolom tabu *active* mengindikasikan berapa banyak iterasi garis yang ada pada kolom tersebut akan berada dalam status tabu. Pada iterasi pertama (iterasi pertama setelah 4-*tree* terbentuk dengan konstruksi *greedy*), tidak ada garis yang berada dalam status tabu aktif, tapi pada iterasi kedua, garis-garis yang digunakan pada satu iterasi sebelumnya (menambahkan (4,6) dan membuang (4,7)) diberi status tabu. Di iterasi berikutnya, garis (4,6) dilepaskan dari status tabu (karena tabu *tenure* hanya satu iterasi), tapi garis (4,7) tetap berada pada status tabu aktif karena tabu *tenure*-nya dua iterasi. Gambar berikut mendeskripsikan bagaimana prosedur penggantian garis (*move*) bekerja.

Perhatikanlah bahwa pada iterasi kedua terdapat *move* yang memberikan yang lebih baik yaitu 49 dengan cara membuang garis (6,7) dan menambahkan garis (4,7). Akan tetapi, karena garis (4,7) dalam tabu *tenure*, dan masih dalam kondisi tabu active, sehingga garis (4,7) terlarang untuk dipilih. Oleh karena itu, pencarian berikutnya mengarah ke garis (6,8) yang akan dimasukkan ke *tree* dan membuang garis (6,7).
Tabel berikut ini menggambarkan proses sampai iterasi ke 9.

Tabel 8. Tabel move (pergantian garis) sampai iterasi ke 9.

Iterasi	*Tabu tenure*		Ditambahkan	Dibuang	Nilai *Move*	Total bobot
	1 iterasi	2 iterasi				
1			(4,6)	(4,7)	7	47
2	(4,6)	(4,7)	(6,8)	(6,7)	10	57
3	(6,8),(4,7)	(6,7)	(8,9)	(1,2)	6	63
4	(6,7),(8,9)	(1,2)	(4,7)	(1,4)	-17	46
5	(1,2),(4,7)	(1,4)	(6,7)	(4,6)	-9	37
6	(1,4),(6,7)	(4,6)	(6,9)	(6,8)	0	37
7	(4,6),(6,9)	(6,8)	(8,1)	(4,7)	1	38
8	(6,8),(8,10)	(4,7)	(9,12)	(6,7)	3	41
9	(4,7),(9,12)	(6,7)	(10,11)	(6,9)	-7	34
10	(6,7),(10,11)	(6,9)	(5,9)	(9,12)	7	41

Seperti telah diilustrasikan pada Tabel 8 proses pencarian terus membangkitkan solusi yang berbeda, dan pada iterasi ke 9 mendapatkan solusi yang terbaik dari solusi-solusi lainnya. Mengaplikasikan Tabu *search* dengan hanya menggunakan *short-term memory* kadang-kadang disebut dengan '*simple Tabu*' atau '*first level Tabu search*'.

Anggap kita ingin mengaplikasikan *diversification strategy* pada masalah ini. Asumsikan juga kita ingin mengulang kembali proses pencarian sampai 6 iterasi sementara kita juga merekam solusi terbaik yang telah didapat.

Karena kita akan melakukan pencarian ulang (*restart*) setelah iterasi ke 6, maka kita akan tetap mencatat solusi awal, dan solusi yang didapat pada iterasi kelima dan keenam akan berada pada status Tabu. Oleh karena itu garis (1,2), (1,4), (4,7),

(6,7, (6,8), (8,9) dan (6,9) berada pada status tabu status. Atribut garis-garis tersebut adalah tabu.

Dengan mensortir kembali garis-garis di graf kecuali garis yang dalam kondisi tabu, dan dengan menggunakan proses pencarian secara *greedy*, maka didapat solusi awal lainnya setelah proses pencarian ulang (*restarting*) sebagaimana ditunjukkan pada tabel berikut:

Tabel 9. Tabel pencarian ulang.

Iterasi	Kandidat yang akan dipilih	Garis yang dipilih	Total bobot
1	(3,5)	(3,5)	6
2	(2,3),(3,4),(3,6),(5,6),(5,9),(5,12)	(5,9)	22
3	(2,3),(3,4),(3,6),(5,6),(5,12),(6,9),(8,9),(9,12)	(8,9)	29
4	(2,3),(3,4),(3,6),(5,6),(5,12),(6,8),(6,9), (7,8), (8,10),(9,12)	(8,10)	38

Setelah proses pencarian ulang (restarting), 38 adalah solusi awal. Garis yang dipilih adalah (3,5), (5,9), (8,9) dan (8,10) yang membentuk solusi awal. Dengan menggunakan cara yang sama didapat tabel sebagai berikut:

Tabel 10. Empat iterasi setelah *restarting*

Iterasi	Tabu *tenure*		Ditambahkan	Dibuang	Nilai *Move*	Total bobot
	1 iterasi	2 iterasi				
1			(9,12)	(3,5)	3	41
2	(9,12)	(3,5)	(10,11)	(5,9)	-7	34 (*)
3	(3,5), (10,11)	(5,9)	(6,8)	(9,12)	7	41
4	(5,9), (6,8)	(9,12)	(6,7)	(10,11)	-3	38

Dari Tabel 10 dapat dilihat bahwa solusi terbaik didapat hanya dengan dua iterasi setelah *restarting* dimana dalam hal ini dalam proses restarting kita menggunakan *neighborhood* (lingkungan) yang berbeda serta solusi awal yang berbeda dengan solusi awal yang didapat pada *first level Tabu Search.*

3.3 Desain Jaringan (*network design*)

Desain jaringan sebagai salah satu bidang optimasi kombinatorial, memainkan peran penting dalam banyak aplikasi kehidupan nyata. Di zaman modern ini di mana model yang akurat dan teknik solusi yang efisien diperlukan, desain jaringan memberikan representasi masalah yang dihadapi. Beberapa contoh desain jaringan meliputi: jaringan transportasi untuk pergerakan komoditas; jaringan komunikasi untuk transmisi informasi; sistem multiprosesor yang kuat untuk menyelesaikan masalah rumit seperti pemrosesan sinyal radar dan banyak lagi.

Secara luas, jaringan terdiri dari dua atau lebih objek yang berbagi sumber daya dan informasi. Berbicara tentang jaringan (*networks*), banyak orang yang menyangka bahwa jaringan tersebut adalah jaringan komputer. Memang betul jaringan komputer adalah salah satu bentuk jaringan, akan tetapi jaringan itu sendiri bukanlah hanya jaringan komputer. Contoh-contoh jaringan yang lain misalnya jaringan listrik, jaringan telekomunikasi, jaringan pipa air bersih, jaringan rute pesawat udara, dan banyak lagi lainnya. Selain itu, terdapat juga istilah jaringan dalam Biologi, yaitu sekumpulan sel yang mempunyai bentuk dan fungsi yang sama. Akan tetapi, pada buku ini bukan jaringan dalam pengertian Biologi ini yang akan didiskusikan, melainkan jaringan seperti yang telah disebutkan sebelumnya.

Menurut kamus bahasa Inggris Cambridge, *network* (jaringan) adalah suatu sistem yang besar terdiri atas banyak bagian yang sama yang saling terhubung agar terjadi komunikasi atau perpindahan antar bagian atau pusat pengontrol. Optimisasi jaringan merupakan masalah yang sangat banyak terapannya dan sangat cepat berkembang saat ini. Dalam optimisasi jaringan, proses desain jaringan itu sendiri memegang peranan yang penting. Sebagai contoh, jika suatu perusahaan ingin membangun suatu jaringan, katakanlah jaringan air bersih di suatu kota, maka yang akan dilakukan adalah memetakan dulu topografi dari kota tersebut dan menyatakan/merepresentasikan tempat-tempat yang akan dihubungkan dengan noktah-noktah atau titik-titik untuk dianalisis sebelum proses instalasi yang sebenarnya dilakukan. Proses merepresentasikan tempat-tempat tersebut menjadi titik-titik dan kemudian dianalisis merupakan salah satu bagian dari desain jaringan.

Pada saat ini dimana model-model yang akurat dan teknik-teknik penyelesaian masalah yang efisien sangat diperlukan, desain jaringan memberikan suatu gambaran yang cepat dan tepat terhadap masalah yang dihadapi. Beberapa contoh dari desain jaringan antara lain: jaringan transportasi untuk pengiriman komoditas, jaringan komunikasi untuk transmisi informasi, dan desain sistem multiprosesor untuk menyelesaikan masalah proses sinyal radar atau sensor, dan sebagainya.

Suatu jaringan adalah suatu sistem yang memuat pergerakan atau aliran dari suatu komoditas seperti produk, air, informasi, arus listrik, panas, surat, manusia, mobil, kereta api, dan sebagainya. Dengan menggunakan koneksi yang ada di jaringan, komoditas biasanya berasal dari sumber dan

berpindah ke terminal/tujuan. Contoh dari jaringan ini adalah jaringan air bersih (PDAM) yang mengalirkan air bersih dari reservoir ke gedung-gedung/rumah melalui pipa-pipa transmissi ataupun pipa-pipa distribusi. Reservoir, gedung-gedung atau rumah-rumah direpresentasikan dengan titik-titik, sedangkan pipa-pipa transmisi maupun distribusi direpresentasikan dengan garis. Contoh lain adalah jaringan transportasi dimana pada jaringan ini titik-titik merepresentasikan kota/stasiun/bandara dan lain-lain dan garis-garis merepresentasikan jalan/rel kereta api/rute-rute penerbangan, dan sebagainya. Jaringan telekomunikasi, listrik, komputer dan lain-lainpun dapat direpresentasikan dengan cara yang sama.

Konsep Teori Graf telah dikenal berguna untuk mempelajari masalah-masalah yang muncul dalam desain dan analisis jaringan. Umumnya struktur graf digunakan untuk memodelkan masalah di sistem jaringan. Sewaktu mendesain suatu jaringan, ada beberapa informasi nonstruktural yang diperlukan dalam jaringan yang harus dipertimbangkan seperti: biaya, kapasitas, jarak, reliabilitas/keterhandalan, waktu, kepadatan lalu-lintas, dan lain-lain. Dengan memberikan bobot baik kepada titik maupun garis di graf yang mewakili jaringan tersebut maka faktor-faktor ini dapat dipertimbangkan dalam model graf.

Pada umumnya titik digunakan untuk merepresentasikan komponen atau objek (stasiun, kota, komputer, depot, dan lain-lain) dan garis merepresentasikan hubungan atau interrelasi antar komponen/objek (rel kereta api, jalan, kabel, dan lain-lain).

BAB IV

BEBERAPA MASALAH YANG MENGGUNAKAN *MINIMUM SPANNING TREE* SEBAGAI *BACKBONE*

Konsep teori graf penting dalam mempelajari masalah-masalah dalam desain jaringan, antara lain: desain suatu jaringan dengan biaya minimal, desain sirkuit terpadu, desain komunikasi data yang mampu untuk mendukung aplikasi *online* dengan skala besar, desain multiprosesor yang canggih untuk menyelesaikan masalah-masalah yang kompleks secara *real-time*, menentukan solusi optimal dalam rute yang diberikan, dan lain sebagainya. Untuk menyelesaikan masalah-masalah desain jaringan ini diperlukan algoritma-algoritma yang selain benar, juga harus efisien. Pengembangan algoritma yang efisien ini dimotivasi oleh beberapa hal, antara lain:

a. Masalah kehidupan nyata yang dapat dimodelkan dalam bentuk desain jaringan dan dirumuskan dengan menggunakan konsep matematika;
b. Model dalam desain jaringan lebih informatif, intuitif, menarik, lebih mudah divisualisasikan, serta lebih mudah dimengerti.

Kedua alasan tersebut di atas membuat masalah-masalah desain jaringan lebih mungkin untuk digunakan dan diimplementasikan. Sehingga, diperlukan adanya algoritma yang benar dan efisien untuk menyelesaikan masalah desain jaringan.

Masalah *Minimum Spanning Tree* (MST) adalah salah satu masalah klasik yang timbul dalam banyak aplikasi desain jaringan. Masalah dasarnya adalah membangun suatu jaringan dengan biaya minimum yang memenuhi parameter yang ditentukan. Parameter yang dipertimbangkan dalam desain jaringan meliputi: jarak, diameter, derajat, konektivitas, reliabilitas/keterhandalan, aliran, dll. Sebagai contoh, dalam jaringan transportasi pembatasan jarak pada barang (aliran komoditas) dapat mewakili jarak maksimum yang diizinkan untuk pengiriman barang/komoditas, dan dalam jaringan radio pembatasan diameter pada stasiun dapat mewakili jumlah maksimum *link*/tautan yang diizinkan untuk berkomunikasi antar studio.

Secara umum MST banyak digunakan sebagai *backbone* atau struktur utama pada banyak masalah desain dan optimisasi jaringan. Sejak G.R Kirchoff mendesain sirkuit listrik dengan menggunakan MST pada abad ke 19, MST telah dianggap sebagai salah satu bentuk graf yang banyak digunakan untuk aplikasi desain jaringan oleh para ilmuan. Deo dan Kumar (1997) menyatakan bahwa pada suatu graf terhubung berbobot, maka untuk menentukan *spanning tree* dari graf tersebut dapat dihitung dangan fungsi waktu yang linear, sedangkan jika *spanning tree* tersebut harus memiliki bobot yang minimum, maka waktu komputasi mengalami sedikit peningkatan.

Algoritma untuk menentukan MST adalah algoritma waktu polinomial, yang berarti bahwa setiap kasus dari masalah MST dapat diselesaikan dalam waktu polinomial. Namun, jika suatu MST diberi kendala tambahan, maka kendala tambahan ini sering membuat masalah menjadi 'sulit' (tidak waktu polinomial lagi). Misalnya, suatu MST yang diberi kendala tambahan seperti derajat (*degree*) atau diameter, akan membuat masalah MST tersebut secara komputasi menjadi 'sulit', dan pada kenyataannya (terlepas dari kasus-kasus yang trivial), masalah itu adalah masalah *NP-complete atau NP-Hard.*

4.1. Beberapa Masalah dengan MST sebagai *Backbone*.

Diberikan graf G(V,E), V= {1, 2, ..., n}, E={e_{ij} | i, j $\in$ V}, c_{ij}>=0, dengan c_{ij} adalah bobot dari garis e_{ij}, beberapa masalah yang menggunakan MST sebagai struktur utama atau *backbone* dari permasalahannya antara lain:

a. Degree Constrained Minimum Spanning Tree (DCMST) Problem

Masalah DCMST dapat dinyatakan sebagai berikut:
Diberikan graf G=(V,E) dan bilangan bulat positif b_1, b_2,...,b_n, tentukan suatu MST T dari G sehingga derajat (*degree*) d_i dari setiap titik i di T paling banyak b_i, $1 \leq i \leq n$.

b. Capacitated Minimum Spanning Tree (CMST) Problem

Masalah CMST dapat dinyatakan sebagai berikut:
Diberikan graf tak berarah G(V,E), dengan titik v_0 sebagai *root* (akar). Selain c_{ij}, setiap garis e_{ij} mempunyai kapasitas s_{ij} dan persyaratan titik r. Tentukan suatu MST yang memenuhi

kendala kapasitas. Kendala kapasitas adalah $s_{ij} \geq \sum_{u \in U(e_{ij})} r(u)$ dengan $U(e_{ij})$ adalah himpunan titik-titik yang mempunyai lintasan di T ke *root* v_0 yang memuat e_{ij}.

c. *Bounded Diameter Minimum Spanning Tree (BDMST) problem*

Masalah BDCMST dapat dinyatakan sebagai berikut:
Diberikan suatu graf berbobot tak berarah G(V,E), dan bilangan positif D, tentukan MST yang memuat diameter ≤ D.

d. k-th Best Minimum Spanning Tree

Diberikan graf tak berarah berbobot G(V,E), dan bilangan bulat positif k, tentukan suatu *spanning tree* T sehingga tepat ada sebanyak (k-1) *spanning tree* yang berbeda, yang tiap *spanning tree* tersebut *mempunyai* bobot lebih kecil dari T .

e. Quadratic Minimum Spanning Tree (QMST) Problem

Diberikan graf tak berarah berbobot G(V,E), dengan bobot c_{ij} untuk tiap garis e_{ij},, dan biaya antara $b\{e_{ij},e_{mn}\}$ untuk tiap pasangan titik $\{e_{ij},e_{mn}\}$, dengan e_{ij}, $e_{mn} \in E$. Tentukan *spanning tree* T sehingga *quadratic cost* $\sum_{e_{ij} \in T} c_{ij} + \sum_{e_{ij}, e_{mn} \in T} b\{e_{ij}, e_{mn}\}$ adalah yang terkecil dari semua *spanning tree* dari G.

f. *Most-Reliable Minimum Spanning* Tree

Diberikan graf tak berarah berbobot G(V,E), dan sebuah fungsi $p: 2^v \rightarrow \{0,1\}$yang menunjukkan peluang dari adanya himpunan bagian dari titik-titik. Tentukan suatu *spanning tree* T

sehingga ekspektasi bobot $E[W_T] = \sum_{S \subseteq V} p(S) W_T(S)$ adalah yang terkecil dari semua *spanning tree* dari G. $W_T(S)$ adalah bobot dari *subtree* yang didapat dengan mempertahankan hanya garis-garis di T yang tepat diperlukan untuk menghubungkan himpunan S.

Dari berbagai bentuk masalah yang menggunakan MST sebagai *backbone*, pada buku ini akan didiskusikan tentang *Degree Constrained Minimum Spanning Tree* (DCMST) problem.

4.2 *Degree constrained minimum spanning tree*

Masalah *Degree Constrained Minimum Spanning Tree* (DCMST) adalah masalah menentukan MST dari suatu graf tetapi juga memenuhi syarat keterbatasan derajat (*degree*) pada setiap titiknya. Bentuk terapan dari DCMST ini umumnya muncul pada masalah yang berhubungan dengan keterhandalan jaringan, misalnya adalah masalah desain jaringan telekomunikasi, jaringan transportasi, jaringan distribusi. DCMST juga digunakan sebagai subproblem pada masalah desain jaringan komputer.

Pada jaringan transportasi, DCMST digunakan untuk mendesain jaringan sedemikian sehingga jaringan transportasi tersebut dapat menghubungkan sejumlah jalan yang menghubungkan sejumlah kota, akan tetapi mempunyai keterbatasan banyaknya jalan yang dapat bertemu di suatu persimpangan. Untuk jaringan komputer, batasan derajat (*degree*) digunakan untuk membatasi banyaknya interkoneksi pada komputer, dan untuk jaringan pipa air bersih, maka kendala *degree* ini digunakan untuk membatasi banyaknya

sambungan pada satu titik di pipa transmisi ataupun di pipa distribusi.

DCMST dapat dimodelkan sebagai bentuk *Mixed Integer Linear Programming* (MILP) berikut ini:

$\text{Min} \quad \sum_i^n \sum_j^n c_{ij}\, x_{ij}$ (1)

kendala

$\sum_{i,j} x_{ij} = n - 1$ (2)

$\sum_{i,j \in V'} x_{ij} \leq |V'| - 1, \ \forall \emptyset \neq V' \subseteq V$ (3)

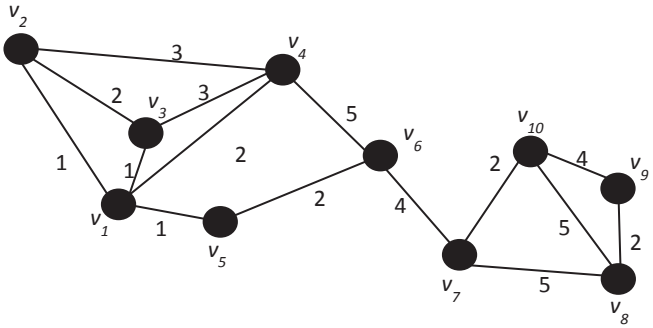
$1 \leq \sum_{j=1, i \neq j} x_{ij} \leq b_i \ , i = 1,2,\ldots, n.$ (4)

$x_{ij} = 0$ atau 1, $\quad 1 \leq i \neq j \leq n.$ (5)

c_{ij} adalah bobot dari garis e_{ij}, b_i adalah batas atas derajat titik i dan n adalah banyak titik di graf G. Kendala (2) memastikan bahwa (n-1) garis dipilih, kendala (3) merupakan kendala yang mengeliminasi terjadinya *cycle*, kendala (4) adalah kendala dari tiap titik di G, dimana pada akhir proses, setiap titik di G harus ada pada *spanning tree* yang terbentuk dan juga memenuhi batas *degree* tiap titik. Kendala (5) merupakan kendala variabel yang membatasi nilai variabel yaitu bernilai 0 atau 1. Jika x_{ij} bernilai 1 maka garis x_{ij} dipilih atau berada di T, sebaliknya, jika x_{ij} bernilai 0 maka x_{ij} tidak berada di T. Formulasi (model) ini merupakan model yang biasa atau umum digunakan untuk menyelesaikan masalah DCMST.

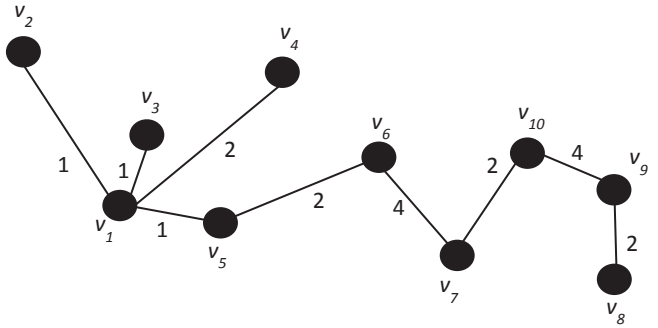
Banyak peneliti yang telah menginvestigasi masalah DCMST dan mengembangkan metode/algoritma untuk menyelesaikannya, baik metode *exact* maupun *heuristics*.

Berikut ini diberikan contoh masalah DCMST. Misalkan diberikan suatu jaringan yang digambarkan sebagai berikut:



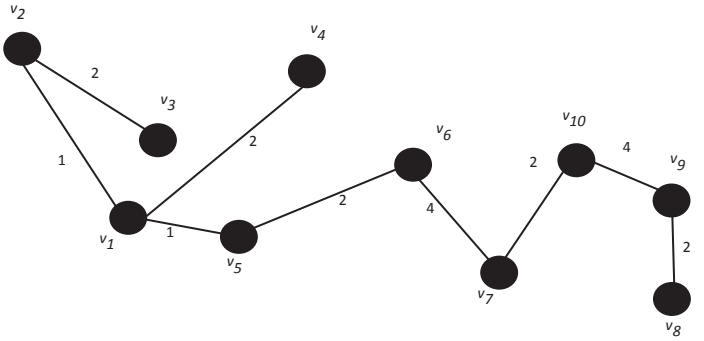
Gambar 48. Contoh jaringan distribusi.

Misalkan titik-titik v_1, v_2, ..., v_{10} adalah titik-titik yang mewakili gedung gedung di suatu fasilitas kesehatan, sedangkan garis-garis yang menghubungkan antar titik adalah rencana panjang pipa-pipa yang menghubungkan antar gedung. Pipa tersebut digunakan untuk mengalirkan aliran gas yang diperlukan pada fasilitas kesehatan tersebut. Jika diinginkan fasilitas tersebut semua terhubung dengan syarat panjang pipa adalah minimum, maka masalah tersebut adalah masalah *Minimum Spanning Tree*, dan jaringan yang terbentuk adalah sebagai berikut:



Gambar 49. MST dari jaringan pada Gambar 48 dengan bobot 19.

Untuk menghindari kemungkian terjadinya kebocoran, misalkan sambungan pipa pada tiap titik (gedung) dibatasi hanya maksimal tiga sambungan, maka masalah tersebut menjadi masalah DCMST dengan batas kendala *degree* 3, dan didapat solusi sebagai berikut:



Gambar 50. DCMST dari jaringan pada Gambar 48 dengan bobot 20

4.3 Metode-metode *Exact* yang telah dikembangkan untuk menyelesaikan DCMST.

Metode *Branch and Bound.*

Metode *Branch and Bound* telah dikembangkan oleh S.C. Narula dan C. A. Ho pada tahun 1980 untuk menyelesaikan masalah DCMST. Metode *Branch and Bound* digunakan bersamaan dengan penalty untuk mengkonstruksi DCMST. Metode yang dikembangkan oleh Narula dan Ho ini menggunakan Algoritma Prim untuk menentukan *Minimum Spanning Tree*. *Minimum spanning tree* ini digunakan sebagai nilai *lower bound.* Untuk menentukan nilai *upper bound* digunakan modifikasi dari Algoritma Prim. Proses pencarian yang dilakukan bergerak untuk menentukan solusi optimal

yang nilainya terletak antara nilai *upper bound* dan nilai *lower bound*.

Untuk proses pencarian (*searching*), metode *Branch and Bound* digunakan dengan mengadopsi metode yang dilakukan oleh Held dan Karp yang dikembangkan pada tahun 1971 untuk menyelesaikan masalah *Travelling Salesman Problem*. Metode yang dikembangkan ini diuji dengan menggunakan 120 data yang merupakan graf dengan orde 30 dan 50 dengan masing masing orde sebanyak 50 data, dan orde 100 dengan sebanyak 20 data. Batas *degree* yang digunakan adalah 3 dan 4. Hasil investigasi mereka menunjukkan bahwa untuk graf dengan orde 30 dan 50 metode tersebut menghasilkan solusi yang optimal (untuk batas *degree* 3 maupun 4), sementara untuk orde 100, untuk batas *degree* 3, hanya 6 dari 20 data/masalah yang menghasilkan solusi yang optimal, sedangkan untuk batas *degree* 4, semua data menghasilkan solusi yang optimal.

Selain Narula dan Ho, Savelbergh dan Volgenant juga menggunakan metode *branch and bound* untuk menyelesaikan DCMST pada tahun 1985 berdasarkan analisis pertukaran garis (*edge exchange analysis*). Metode ini diuji dengan menggunakan data yang sama yang digunakan oleh Narula dan Ho (1980) serta menambahkan 10 data untuk graf dengan orde 70 dan 10 data untuk graf dengan orde 200. Untuk batasan *degree* digunakan batasan *degree* 3 untuk semua data.

Pada metode yang dikembangkan oleh Savelbergh dan Volgenant ini, ada beberapa hal yang menjadi catatan penting sewaktu membentuk MST yaitu:

a. Setiap *branch* dari *tree* tidak lebih panjang dari *chord* yang ada pada *fundamental cutset*.
b. Setiap *chord* dari *tree* tidak lebih panjang dari *branch* yang ada di *fundamental path*.

Metode Relaksasi Lagrange

Seperti telah didiskusikan sebelumnya bahwa beberapa masalah optimisasi kombinatorial menjadi sulit diselesaikan jika masalah tersebut ditambah dengan parameter atau kendala baru dan Metode Relaksasi Lagrange mengatasi masalah tersebut dengan mengangkat kendala yang sulit tersebut menjadi bagian dari nilai fungsi tujuan dengan mengalikannya denga vektor pengali Lagrange dan memberikan penalti.

Gavish pada tahun 1982 dan Volgenant pada tahun 1989 memodifikasi model DCMST dengan mengalikan kendala degree dengan vektor pengali Lagrange π dan kemudian menambahkannya ke fungsi tujuan dan mendapatkan model/formulasi berikut untuk masalah DCMST:

(P_1):

$$Minimize: \sum_{i,j \in V} c_{ij} + \sum_{i=1}^{n} \pi_i \left(\sum_{j=1}^{n} x_{ij} - b_i \right)$$

Kendala: $\sum_{j=1}^{n} x_{ij} \geq 1, \quad 1 \leq i \leq n$

$\pi_i \geq 0, \quad 1 \leq i \geq n$

$\sum_{i,j \in V'} x_{ij} \leq \ | V' | - 1, \quad \forall \emptyset \neq V' \subseteq V$

$x_{ij} = 0$ atau $1, \qquad 1 \leq i \neq j \leq n.$

Berdasarkan Teorema 1 dari Geoffrion pada tahun 1974, solusi X dari masalah DCMST:

$Minimise \ \sum_i^n \sum_j^n c_{ij} x_{ij}$

Kendala :

$\sum_{i,j} x_{ij} = n - 1$

$\sum_{i,j \in V'} x_{ij} \leq \ |V'| - 1, \quad \forall \emptyset \neq V' \subseteq V$

$1 \leq \sum_{j=1, i \neq j} x_{ij} \leq b_i \quad i = 1,2,\ldots,n$

$x_{ij} = 0$ or 1, $\quad 1 \leq i \neq j \leq n$.

optimal jika X memenuhi, untuk π yang diberikan, memenuhi tiga kondisi berikut:

(i) X is optimal in P_1

(ii) $\sum_{j=1, i \neq j} x_{ij} \leq b_i \quad i = 1,2,\ldots,n$

(iii) $\pi_i \left[\sum_{j=1}^n x_{ij} - b_i\right] = 0, \quad i = 1,2,\ldots, n.$

Maka P_1 adalah *lower bound* untuk DCMST dan *bound* terbaik untuk tipe ini adalah $P^* = \max_{\pi \geq 0} P_1$.

Kualitas dari *bound* sangat penting pada metode *Branch and Bound*, sehingga Relaksasi Lagrange dapat membantu untuk memperbaiki kualitas *bound*, terutama untuk masalah-masalah yang sulit. Walaupun demikian, menentukan vektor pengali Lagrange bukanlah sesuatu yang mudah.

Volgenant pada tahun 1989 mengembangkan metode Relaksasi Lagrange untuk menyelesaikan DCMST. Pada metode yang dikembangkannya didapat nilai *lower bound* yang lebih baik dengan menggunakan pengali Lagrange. Pada metode Ascent (metode yang dikembangkannya), nilai pengali Lagrange awalnya di set nol, tapi jika *degree* dari titik i yang dipilih melebihi batas *degree* yang diberikan, nilai pengali Lagrange di set > 0. Dari hasil implementasi yang dilakukan menunjukkan nilai solusi yang didapat sangat dekat dengan solusi optimal.

Branch and Cut

Pada metode *Branch and Cut* terdapat tiga prosedur utama yaitu: prosedur reformulasi, *heuristic* dan prosedur *cutting plane*. Ketiga prosedur ini dikombinasikan ke proses pencarian pada metode *branch and bound*.

Caccetta and Hill pada tahun 2001 mengganti kendala (2) dan (4) dengan kendala (6), (7), dan (8) berikut karena masalah kemudahan komputasi untuk menyelesaikan DCMST dengan menggunakan metode *Branch and Cut*:

$$\sum_{j \in V} x_{ij} - d_i = 0, \quad 1 \leq i \leq n \qquad (6)$$

$$\sum_{i=1}^{n} d_i = 2(n-1) \qquad (7)$$

$$1 \leq d_i \leq b_i, \quad 1 \leq i \leq n \qquad (8)$$

Bentuk formulasi ini mempunyai tambahan n variabel (d_i dengan i=1,2,...n), akan tetapi banyaknya kendala tereduksi sebanyak n.

Formulasi yang digunakan oleh Caccetta dan Hill ini digunakan untuk menyelesaikan DCMST dengan metode *branch and cut*. Nilai *upper bound* dibangkitkan dengan menggunakan *heuristic* yang dikembangkan oleh Savelsbergh dan Volgenant (1985). Nilai *lower bound* dibangkitkan dengan menggunakan Relaksasi Lagrange yang digunakan oleh Volgenant (1989). Hal yang penting dalam metode *branch and cut* yang mereka kembangkan adalah: penggunaan teknik DFS, penggunaan paket program CPLEX untuk menyelesaikan Program Linear relaksasi.

4.4 Metode-metode *Heuristic* yang telah dikembangkan untuk menyelesaikan DCMST

Metode *exact* memiliki kelebihan dan kekurangan. Keuntungan utama adalah metode ini memberikan solusi yang optimal. Namun, untuk digunakan dalam masalah skala besar, mungkin tidak begitu efektif karena memakan waktu komputasi, dan hal ini adalah kelemahan utamanya. Oleh karena itu, prosedur *heuristic* atau aproksimasi yang efektif menjadi menarik untuk digunakan.

Jika metode *exact* memberikan solusi yang optimal, metode *heuristic*, di sisi lain, menawarkan solusi yang biasanya mendekati optimal atau sub-optimal. Seperti yang telah dikemukakan pada bab sebelumnya, meskipun solusi yang diperoleh dengan menerapkan metode *heuristic* tidak dapat dijamin optimal, namun dapat memberikan solusi layak yang baik dalam waktu yang dapat diterima secara wajar dan relatif cepat.

Untuk menyelesaikan masalah DCMST, banyak *heuristic* yang telah diselidiki, antara lain algoritma *greedy*, algoritma genetika, Tabu *Search* dan lain-lain.

Algoritma *Greedy*

Dua metode *greedy* yang terkenal untuk menentkan MST adalah Algoritma Prim dan Algoritma Kruskal. Algoritma Kruskal mengkonstruksi suatu *spanning tree* dengan cara terus menerus memilih garis terkecil pada suatu graf dengan syarat penambahan garis tidak menyebabkan terjadinya *cycle*.

Algoritma Prim berbeda dengan Algoritma Kruskal dalam proses konstruksi *tree*. Pada Algoritma Kruskal dimungkinkan terbentuknya *forest* pada proses konstruksi, walaupun pada akhirnya semua titik akan terhubung sewaktu proses

konstruksi selesai. Berbeda dengan Algoritma Kruskal, pada Algoritma Prim waktu proses konstruksi berlangsung tetap menjaga keterhubungan dari graf, sehingga tidak adan terbentuk *forest* pada waktu proses konstruksi.

Narula dan Ho pada tahun 1980 mengembangkan dua *heuristic* untuk menyelesaikan DCMST. *Heuristic* pertama (disebut dengan Primal *heuristic*) adalah modifikasi dari Algoritma Prim. Pada Primal *heuristic* ini solusi yang didapat merupakan solusi yang layak karena Algoritma Prim untuk menentukan MST tersebut dimofikikasi untuk mengakomodir kendala *degree*. *Heuristic* kedua (disebut dengan Dual *heuristic*) adalah *heuristic* yang dimulai dengan MST yang dibangkitkan dari Algoritma Prim yang kemudian bergerak menuju layak melalui sejumlah pertukaran garis Algoritma yang dikembangkan oleh Narula dan Ho ini diuji dengan data yang sama yang mereka gunakan untuk menguji metode *branch and bound* yang mereka kembangkan. Kualitas Primal dan Dual *Heuristic* berada 1,6% dari *lower bound*. Primal *heuristic* menghasilkan solusi awal yang layak dalam waktu yang lebih cepat dibandingkan Dual *heuristic*.

Iterative Refinement

Boldon dkk. Pada tahun 1996, serta Deo dan Kumar pada tahun 1997 mengembangkan metode *Iterative Refinement*. Mereka mengembangkan 4 versi dari metode tersebut yang diberi nama Algoritma BF1, BF2, BF3 dan BF4. Semua algoritma dimulai dengan menentukan MST dan kemudian garis yang menempel pada titik yang melangar kendala *degree* akan diberikan penalty. Perbedaan dari ke 4 macam metode yang mereka kembangkan terletak pada banyaknya garis yang diberikan penalty dan fungsi penalty itu sendiri.

Pada metode BF1 dan BF3, semua garis yang menempel dengan titik yang melanggar kendala *degree* diberi penalty, sedangkan untuk metode BF2 dan BF4 semua garis yang menempel dengan titik yang melanggar kendala *degree* diberi penalty, kecuali garis dengan bobot terkecil. Dengan kata lain, pada metode BF2 dan BF4, satu garis dengan bobot terkecil tidak diberikan penalty. Dengan bobot baru tersebut, proses untuk menghitung MST dilakukan kembali. Proses itni dilakukan terus menerus sampai didapat suatu *spanning tree* yang tidak memuat titik yang melanggar kendala *degree* yang berarti dalam hal ini telah didapat DCMST.

Keempat algoritma yang mereka kembangkan diimplementasikan dengan menggunakan komputasi paralel yang menggunakan 8192 processor. Diantara keempat algoritma yang telah dikembangkan, BF2 merupakan algoritma yang mempunyai kinerja terbaik. Selain itu, data yang diimplentasikan memuat graf dengan orde 5934 (memuat 5934 titik). Walaupun demikian, kualitas dari solusi dengan pendekatan ini tidak terlalu baik. Mereka melaporkan bahwa untuk menyelesaikan DCMST dengan batasan *degree* 5, solusi yang dihasilkan terletak sekitar 40% sampai 70% dari *lower bound* (MST).

Tabu Search

Algoritma Tabu *search* untuk menyelesaikan DCMST telah dikembangkan oleh Wamiliana dan Caccetta pada tahun 2003. Mereka mengembangkan tiga metode berdasarkan Tabu *search* yang diberi nama CW1, CW2 dan CW3. Metode Tabu *search* mulai proses sama dengan pencarian lokal (l*ocal search*) atau pencarian lingkungan (*neighbourhood search*). Proses pencarian dilakukan secara iterative dari satu solusi ke solusi

lainnya sampai kriteria penghentian tercapai. Setiap solusi didapat dari solusi sebelumnya dengan suatu operasi yang disebut dengan *move*.

Pada metode yang mereka kembangkan digunakan beberapa istilah yaitu: G adalah graf terhubung berbobot, T adalah MST yang dibangkitkan dari G, *Tabu move* adalah himpunan garis yang diberikan status tabu aktif dan akan berada dalam kondisi tersebut dalam beberapa iterasi, *tabu tenure* adalah banyaknya iterasi yang menjaga suatu solusi dalam kondisi tabu, dan *move* adalah himpunan garis yang menempel dengan daun di G\T.

Algoritma mulai dengan menentukan MST yang merupakan *lower bound* (LB), sedangkan untuk menentukan *upper bound* (UB) digunakan *Degree Constrained Spanning Tree* (DCST) yang didapat dengan menggunakan modifikasi dari Algoritma Kruskal. Jika nilai DCST sama dengan nilai MST, maka proses selesai, solusi optimal didapat, solusi MST = solusi DCST = solusi DCMST. Jika tidak, maka pencarian solusi dimulai dari nilai *upper bound* (yang merupakan solusi layak) yang bergerak menuju optimal. *Moves* adalah himpunan garis yang menempel dengan daun (titik yang berderajat 1) di G\T. Tabu *tenure* di set 0,1 n dengan n adalah orde (banyaknya titik) dari graf. Maksimum banyaknya iterasi adalah 20% dari banyaknya titik, sedangkan kriteria penghentian adalah maksimum banyaknya iterasi dan *tolerance*, dimana t*olerance* = 1% dari *gap*, dan *gap* = UB − LB. Perhatikanlah bahwa nilai UB akan direvisi Ketika solusi layak yang lebih baik didapat.

Kriteria aspirasi diaplikasikan jika sewaktu proses *move*, dideteksi adanya pelanggaran kendala batas *degree* dari titik *i*. Semua pertukaran garis yang mungkin di T yang menempel dengan titik *i*, dan garis-garis di G yang tidak ada di T yang

merupakan lingkungan (*neighbour*) dari I, akan diperiksa. Jika proses pencarian tidak mendapatkan solusi yang lebih baik, maka solusi terbaik yang telah didapat direkam sebagai solusi terbaik saat ini x^*, lalu masukkan garis yang baru digunakan dalam proses move ke tabu status dan proses pencarian diulang kembali. Berikut diberikan bentuk *pseudocode* dari algoritma ini:

```
begin
   set t=0
   initialize: graph, Tabu_move, Tabu_status
   find MST, DCST
   set control T: DCST
   while (not termination condition)
   do
        choose the best move
        perform edge exchange
        if (new solution feasible)
        compare the new solution with control T
        adjust the Tabu_move and Tabu_move
        set t: t+1
       endif
    end
end
```

Algoritma Tabu ini diimplementasikan dengan menggunakan 2160 data yang dibangkitkan dengan kriteria sebagai berikut:

- Orde graf adalah 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 200, 250, 300, 350, 400, 450, 500.

- Bobot tiap garis bernilai integer dan dibangkitkan secara random dengan menggunakan distribusi uniform (1, 1000).
- Untuk tiap orde dibangkitkan 30 data.
- Untuk tiap n, graf dibangkitkan dengan nilai p yang berbeda, p adalah peluang suatu garis ada di graf. Digunakan p=0.25, 0.5, 0.75 dan 1. Untuk p=1 merupakan graf lengkap dimana semua pasangan titik ada garis yang menghubungkannya.

  Untuk nilai p yang diberikan, garis e_{ij} dipilih untuk berada di graf jika untuk q (random) yang dibangkitkan, 0 < q< 0 , nilai q < p.

  Ekspektasi banyaknya garis di graf adalah $\binom{n}{2}$ p.

  Untuk nilai p selain 1 dimungkinkan bahwa graf yang terbentuk adalah graf-graf yang tak terhubung. Jika hal ini terjadi, maka graf tersebut dibuang dan dibangkitkan lagi graf baru yang merupakan graf terhubung.

Dari hasil implementasi terhadap 210 data dengan batas kendala degree 3, dengan menggunakan nilai rasio $\frac{H-LB}{LB}$ menghasilkan rata-rata 6,5% dari LB.

Selain diimpelentasikan terhadap 2160 data, metode ini juga diuji dengan menggunakan data yang diambil dari TSPLIB di http://www.iwr.uniheidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB yaitu data *benchmark* pr264, att532, dan rat575.

Tabel berikut memberikan perbandingan solusi beberapa metode yang dikembangkan untuk menyelesaikan DCMST dengan menggunakan data *benchmark* tersebut.

Tabel 11. Perbandingan beberapa metode untuk menyelesaikan DCMST dengan data *benchmark*.

Algoritma	pr264	att532	rat575
MST	41142	75872	6246
BF2 (Deo, Kumar)	41143	NA	6265
BF4 (Deo, Kumar)	41143	NA	6265
GA-F (Krishnamoorthy et al)	41142	75981	6250
GA-P (Krishnamoorthy et al)	44344	75981	6397
PSS (Krishnamoorthy et al)	41143	75981	6250
SA (Krishnamoorthy et al)	43438	79046	6393
BB (Krishnamoorthy et al)	41143	75912	6250
BB (Volgenant)	41143	75912	6250
Modified Kruskal	41155	75968	6265
Modified Prim	41143	75991	6263
Tabu (CW2)	41143	75968	6263
Tabu (CW3)	41145	75954	6265

Metode *Modified Penalty*

Metode *Modified Penalty* dikembangkan oleh Wamiliana pada tahun 2004. Metode ini dikembangkan berdasarkan ide dari metode *Iterative Refinement* yang dikembangkan oleh Boldon dkk. pada tahun 1996 serta Deo dan Kumar pada tahun 1997. Modifikasi dilakukan karena metode tersebut diimplementasikan dengan menggunakan komputer yang hanya mempunyai satu prosesor. Terdapat dua variasi dari metode ini yang disebut dengan MP1 dan MP2. Modifikasi utama yang dilakukan adalah:

1. Untuk MP1, garis dengan bobot terkecil yang menempel dengan titik yang melanggar kendala batas *degree* tidak dipenalty, dan untuk MP2 sebanyak $b_i - 1$ garis yang terkecil

yang menempel dengan titik yang melanggar kendala *degree* tidak dikenakan penalty.
2. Proses penalty dilakukan secara sekuensial, dan tidak ada prioritas terhadap garis garis yang di penalty. Titik-titik yang melanggar kendala batas *degree* ditangani satu demi satu berdasarkan indeks terkecil.

Perhatikahlah disini bahwa MP1 mirip dengan BF2, akan tetapi jika BF2 menggunakan komputasi paralel sedangkan MP1 diimplentasikan dengan menggunakan prosesor tunggal. Pada metode BF2 proses penalty dilakukan secara simultan untuk semua titik tang melanggar kendala batas *degree*, sedangkan untuk MP1 dilakukan secara sekuensial.

Berikut ini diberikan Algoritma MP1:

Langkah 1:
Tentukan MST T, cek apakah ada batas *degree* yang dilanggar, jika tidak ada STOP, solusi optimal didapat. Jika ada, masukkan titik yang melanggar kendala *degree* pada himpunan D. Lanjutkan ke Langkah 2.

Langkah 2:
Definisikan $W_{\min}$ sebagai garis dengan bobot terkecil di T dan $W_{\max}$ sebagai garis dengan bobot terbesar di T.

Langkah 3:
Pilih satu elemen di D, misalkan elemen tersebut i, dan definisikan:

$$f(e_{ij} = \begin{cases} 1, \text{jika } d_i < b_i \text{ dan } d_j > b_j \text{ atau } d_i > b_i \text{ dan } d_j < b_j \\ 2\text{ , jika } d_i > b_i \text{ dan } d_j > b_j \end{cases}$$

dengan d_i = *degree* titik i

b_i = *batas* dari *degree* titik i.

Langkah 4:
Untuk setiap garis yang menempel dengan i (kecuali garis dengan bobot terkecil) hitunglah bobot garis yang baru dengan menggunakan rumus:

$$W'(e_{ij}) = W(e_{ij}) + kf(e_{ij})\left(\frac{W(e_{ij})}{W_{\max} - W_{\min}}\right)W_{\max} \qquad (9)$$

Langkah 5:
Sortir ulang garis dan kembali ke Langkah 1.

Algoritma MP2 mirip dengan Algoritma MP1. Perbedaan utama terletak pada Langkah 4 dimana pada MP1, satu garis dengan bobot terkecil yang menempel pada titik yang melanggar kendala *degree* tidak dikenakan penalty, sementara di MP2 sebanyak b_i - 1 garis yang tidak dipenalty. Pada langkah ini banyaknya garis yang dipenalty yang menempel pada titik i yang melanggar kendala batas pada MP2 adalah sebanyak d_i – (b_i - 1). Sehingga, jika *degree* dari titik i adalah 5 dan batas *degree* dari titik tersebut 3, maka banyaknya garis yang menempel pada titik i yang dipenalty, jika menggunakan MP1 ada sebanyak 4 garis, sedangkan di MP2 sebanyak 3.

Terdapat tiga komponen yang menentukan kualitas solusi dari metode Modified Penalty yaitu *combing factor k*, nilai *f* dan rasio $\left(\frac{W(e_{ij}) - W_{\min}}{W_{\max} - W_{\min}}\right) W_{\max}$.

Combing factor k di rumus penalty seperti tercantum pada persamaan (9) pada Langkah 4 memainkan peranan yang penting. Memilih nilai *k* dekat dengan 1 kadang-kadang menghasilkan solusi yang baik, tapi umumnya dengan memberikan nilai *k* = 1 solusi yang didapat lebih buruk daripada jika nilai nya lebih kecil. Sebaliknya, memilih nilai *k* terlalu kecil, dekat dengan 0 akan menyebabkan lambatnya mendapatkan solusi optimal.

Komponen lain yang mempengaruhi kualitas solusi adlah rasio $\left(\frac{W(e_{ij}) - W\min}{W_{\max} - W_{\min}}\right) W_{\max}$ yang terdapat di persamaan (9) pada Langkah 4. Rasio ini juga menentukan kekonvergensian dari solusi. Rasio yang kecil melambatkan proses penentuan solusi yang layak, akan tetapi terlalu besar rasio akan menyebabkan kualitas solusi yang terlalu jauh dari optimal. *Pseudocode* berikut memberikan gambaran tentang Algoritma *Modified Penalty*.

Input: graph G, constraints C and D, where is the minimum weight constraints and D is the degree constraints

begin

In graph G find a spanning tree that satisfies C

while (spanning tree violates D)

using D alter the weight of edges in G by penalty function to obtain graph G′ with new weights

In graph G′ find a spanning tree that satisfies C

Set G: G′

end while

end

Kinerja dari Algoritma *Modified Penalty* sangat tergantung dari fungsi penalty. Dari ketiga komponen yang berpengaruh terhadap nilai solusi, *combing factor* k yang paling berpengaruh. Ketika nilai k=1, rata-rata solusi yang didapat jauh dari optimal, sementara jika k<1 solusi cenderung lebih baik, tetapi diperlukan waktu yang lebih lama (iterasi yang lebih banyak). Dengan menggunakan perhitungan $\frac{H-LB}{LB}$ rata-rata solusi untuk k=1 adalah 16,2242%, dan untuk k -0,5 adalah 11%.

Algoritma Genetika

Penggunaan Algoritma Genetika untuk menyelesaikan DCMST telah dilakukan oleh Zhou dan Gen pada tahun 1997. Bilangan Prufer digunakan pada proses *encoding* dan *decoding tree*. Penggunaan bilangan Prufer mempunyai kelebihan karena pada bilangan Prufer, titik yang berderajat r akan mumcul sebanyak tepat r -1 kali. Pada metode yang dikembangkan oleh Zou dan Gen diadopsi *uniform crossover*

dan *perturbation mutation* sebagai operator genetika. Peluang *crossover* di set 0,5 dan peluang mutasi 0,1. Metode ini diuji dengan menggunakan 5 data dengan orde bervariasi dari 10 sampai 50. Bobot merupakan integer yang diangkitkan dengan menggunakan distribusi uniform (10, 100). Walaupun dilaporkan bahwa kinerja metode sekitar 8% dari solusi optimal, akan tetapi banyaknya data yang digunakan hanya 5 data saja.

Krishnamoorty dkk pada tahun 2001 mengembangkan metode untuk menyelesaikan DCMST berdasarkan Algoritma Genetika yang dinamakan dengan GA-F dan GA-P. Perbedaan kedua metode ini terletak pada solusi awal. Pada GA-P solusi awal yang tidak layak (yang melanggar kendala *degree*) diperbolehkan, sedangkan pada GA_F hanya solusi awal yang layak yang diperbolehkan.

Mutasi yang digunakan pada kedua algoritma adalah 0,05. Kriteria penghentian untuk GA-F adalah $50n/\bar{b}$, dengan $\bar{b}$ adalah rata-rata maksimum *degree* dari titik-titik yang ada di graf, sementara kriteria penghentian untuk GA-P adalah 100 iterasi jika tidak ada lagi solusi yang lebih baik didapat.

DAFTAR PUSTAKA

Boldon, B., N. Deo and Nishit Kumar (1996). Minimum Weight degree-constrained spanning tree problem: Heuristics and Implementation on an SIMD parallel machine, *Parallel Computing* vol. 22, hal..369 –382Deo, Narsingh, 1989. *Graph Theory and Its Application.*

Caccetta,L. and Hill, S.P.(2001). A Branch and Cut method for the Degree Constrained minimum Spanning Tree Problem', *Networks*, vol. 37, hal.74-83.

Deo N. and Nishit Kumar(1997). Computation of Constrained Spanning Trees: A Unified Approach, *Network Optimization* Lecture Notes in Economics and Mathematical Systems, Editor : Panos M. Pardalos, et al , Springer-Verlag, Berlin, Germany, hal. 194 – 220.

Foulds, L.R.(1992). *Graph Theory Applications*. Springer-Verlag, New York, USA

Geoffrion, A.M. (1974). Lagrangean relaxation for integer programming, *Mathematical Programming Study* 2, hal. 82-114.

Graham, R.L., and Hell, P. (1982). On the history of the Minimum Spanning Tree Problem. Bell Lab, Murray Hill, New Jersey.

Glover, F. (1989). Tabu Search –Part I. ORSA *journal on Computing*, hal. 1, hal.190-206.

Glover, F. (1990). Tabu Search – Part II. ORSA *Journal On Computing*, vol. 2, hal. 4-32.

Glover F., and Manuel Laguna (1997). Tabu Search. Kluwer Academic Publishers, Boston-Massachusetts, USA.

Krishnamoorthy,M., A.T. Ernst and Yazid M Sharaila (2001),. Comparison of Algorithms for the Degree Constrained Minimum Spanning Tree, *Journal of Heuristics*, vol. 7, no. 6, pp. 587-611.

Munir, R. (2005). *Matematika Diskrit*. Bandung: Informatika

Narula,S. C., and Cesar A.Ho.(1980). Degree-Constrained Minimum Spanning Tree. *Computer and Operation Research*, vol. 7,pp. 239-249.

Petrica C Pop. (2012). Generalized Network Design Problems, Modeling and Optimization. *De Gruyter Series in Discrete Mathematics and Applications* 1 (Editor Colva M. Roney-Dougal), Walter de Gruyter GmbH, Berlin

Savelsbergh,M., and T. Volgenant.(1985). Edge Exchange in The Degree Constrained Minimum Spanning Tree. *Computer and Operation Research*, Vol. 12, pp. 341-348.

Vasudev, C. (2006). *Graph Theory with Applications*. New Age International Limited (P) Publishers, Bangalore, India

Volgenant, A.(1989). A Lagrangean Approach to The Degree-Constrained Minimum Spanning Tree Problem, *European Journal of Operational Research*, vol. 39, pp.325- 331

Wamiliana and Caccetta (2003). Tabu search based heuristic for the Degree Constrained Minimum Spanning Tree Problem. Prosiding South East Asia Mathematical Society, hal. 133-140

Wamiliana. (2004). Combinatorial Methods for The Degree Constrained Minimum Spanning Tree Problem. Disertasi

Wamiliana. (2004). Solving The Degree Constrained Minimum Spanning Tree Problem Using Tabu and Modified Penalty Search Methods. *Jurnal Teknik Industri* Vol. 6, No. 1, 1 – 9.

INDEKS