Menaksir Solusi *Travelling Salesman Problem* dengan Algoritma *Christofides*

Antonio Setya / 13516002
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
13516002@std.stei.itb.ac.id

***Abstrak*—Persoalan pedagang keliling (*travelling salesman problem*) memberikan sebuah masalah bagaimana cara menentukan rute terpendek yang dilalui seseorang yang ingin mengunjungi beberapa kota namun dia berangkat dan kembali ke kota yang sama lagi. Persoalan ini dapat direpresentasikan sebagai graf lengkap yang berbobot. Mencari solusi yang tepat dari representasi tersebut memerlukan waktu yang lama, maka dikembangkanlah cara untuk menaksir hasil mendekati solusi yang eksak. Salah satu cara yang melakukan hal tersebut adalah algoritma *Christofides*. Dengan hasil yang mendekati solusi eksak namun dengan kebutuhan waktu eksekusi yang lebih sedikit, algoritma *Christofides* ini dapat digunakan sebagai alternatif mencari solusi dari *travelling salesman problem*.**

***Kata Kunci* — graf berbobot, pohon merentang minimum, algoritma *Christofides*, kompleksitas waktu algoritma.**

I. PENDAHULUAN

Kehidupan kita setiap harinya hampir tidak bisa dipisahkan dari teknologi yang ada sekarang. Sebagian besar kegiatan yang kita lakukan dibantu oleh berbagai perangkat, umumnya oleh komputer pribadi dan *smartphone*. Salah satu kegiatan yang juga dibantu oleh perangkat-perangkat tersebut adalah dalam mencari rute/jalan yang terpendek, dari suatu tempat ke tempat lain, namun kembali lagi ke tempat asal. Dalam dunia komputer, persoalan seperti itu disebut sebagai persoalan pedagang keliling (*travelling salesman problem/TSP*).

Persoalan pedagang keliling menggambarkan bagaimana seorang pedagang yang harus berjualan ke beberapa tempat dan kembali lagi ke tempat asalnya dengan menempuh rute seminimal mungkin. Dalam praktiknya, akan cukup sulit untuk mencari solusi yang optimal dari masalah ini, karena untuk mendapatkan jarak terpendek, maka kita perlu membandingkan semua kemungkinan rute yang ada. Kita bisa saja mencoba satu-satu semua kemungkinan rute yang ada kemudian membandingkan rute mana yang terpendek, namun, jika jumlah tempat yang perlu dikunjungi cukup banyak, hal ini menjadi lebih sulit. Maka dari itu, diperlukan cara lain untuk mendapatkan solusi dari persoalan pedagang keliling ini. Sayangnya, sampai sekarang, untuk persoalan ini, belum ada cara yang dapat menyelesaikan *travelling salesman problem* dalam skala besar dan dalam waktu yang cukup singkat.

Karena mencari solusi yang benar-benar terpendek bukanlah hal yang mudah, maka *computer scientists* mulai beralih untuk bukan lagi mencari solusi yang tepat, namun mengembangkan algoritma untuk mencari solusi yang bisa dikatakan cukup dekat dengan solusi aslinya. Maka dari itu, berkembanglah berbagai cara untuk menemukan perkiraan solusi dari persoalan pedagang keliling ini. Salah satu cara yang berkembang adalah algoritma *Christofides*, dimana algoritma tersebut merupakan salah satu algoritma perkiraan yang pertama kali diciptakan.

Maka dari itu, melalui makalah ini, penulis akan membahas bagaimana algoritma *Christofides* bisa dipakai untuk menjadi alternatif mencari solusi/perkiraan solusi dari persoalan pedagang keliling. Dalam makalah ini, topik-topik utama yang akan dibahas adalah tahapan dalam algoritma *Christofides*, perbandingan dengan algoritma lainnya yang menghasilkan solusi eksak dari persoalan pedagang keliling, dan sebagai tambahan, aplikasi dunia nyata dari persoalan pedagang keliling. Tentunya, akan dibahas pula topik-topik yang berkaitan/mendukung topik utama tersebut.

II. DASAR TEORI

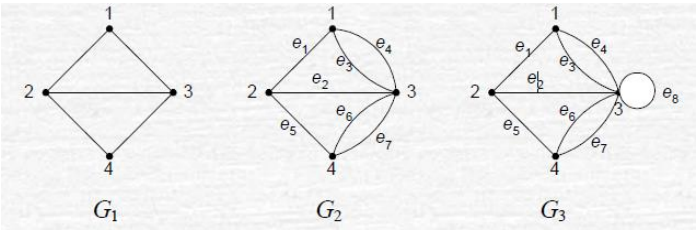
A. Graf

A.1. Definisi Graf dan Jenis Graf

Sebuah graf adalah himpunan dari simpul-simpul/*nodes* yang tidak kosong dan himpunan dari sisi/*edges*. Jika ditulis dalam notasi matematik, misalkan G adalah sebuah graf, maka G adalah tupel dengan dua elemen, yaitu (V,E), dimana V adalah himpunan tidak kosong dari simpul dan E adalah himpunan sisi yang menghubungkan dua simpul yang terdefinisi dalam V.[1]

Dengan definisi seperti diatas, maka kita dapat mengelompokkan berbagai graf menjadi dua berdasarkan ada tidaknya gelang/kalang/*loop* (sisi yang menghubungkan simpul yang sama) dan sisi ganda (dua sisi yang menghubungkan sepasang simpul yang sama),

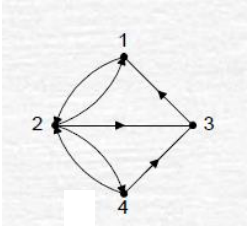
a. Graf sederhana (*simple graph)*, yaitu graf yang tidak mengandung gelang ataupun sisi ganda,
b. Graf tidak sederhana (*unsimple graph),* yaitu graf yang mengandung gelang atau sisi ganda ataupun keduanya. Graf jenis ini dibagi lagi menjadi dua jenis, yaitu graf ganda (*multigraph*), yaitu graf yang memiliki sisi ganda, dan graf semu (*pseudograph*), yaitu graf yang memiliki gelang.



Gambar 1. Contoh (G_1) graf sederhana, (G_2) graf ganda, dan (G_3) graf semu. (Sumber : Rinaldi Munir, Matematika Diskrit rev.ed. 5, hlm. 356)

Berdasarkan sisinya, graf juga bisa dikelompokkan menjadi dua jenis, yaitu,

a. Graf tak berarah (*undirected graph*), yaitu graf yang sisinya tidak memiliki arah, sehingga urutan tidak diperhatikan. Graf-graf pada Gambar 1 adalah contoh dari graf tak berarah.
b. Graf berarah (*directed graph*), yaitu graf yang sisinya memiliki arah, sehingga urutan diperhatikan. [2]



Gambar 2. Contoh graf berarah. (Sumber : Rinaldi Munir, Matematika Diskrit rev.ed. 5, hlm. 359)

A.2. Terminologi Graf

Berikut adalah beberapa terminologi yang sering digunakan dalam mendeskripsikan kondisi sebuah graf dan akan sering digunakan pula dalam makalah ini.

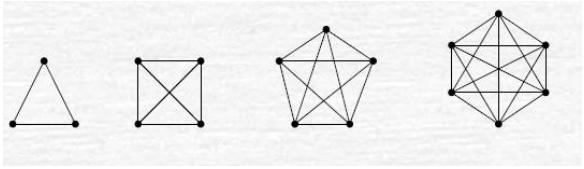
a. Bertetangga (*Adjacent*)
Sebuah simpul dalam graf dikatakan bertetangga dengan simpul lainnya jika kedua simpul tersebut terhubung oleh suatu sisi.
b. Bersisian (*Incident*)
Sebuah sisi bersisian dengan dua buah simpul jika memang sisi tersebut menghubungkan kedua buah simpul tersebut.
c. Derajat (*Degree*)
Derajat sebuah simpul adalah jumlah sisi yang bersisian dengan simpul tersebut. Derajat sebuah simpul *v* ditulis sebagai *deg(v)*. Berkaitan dengan derajat, terdapat suatu kaidah yang disebut dengan kaidah jabat tangan (*handshaking theorem*, sering disebut pula lemma jabat tangan/*handshaking lemma*). Teori ini mengatakan bahwa untuk setiap G = (V,E) yang merupakan graf tak berarah, dengan *n* buah sisi, berlaku persamaan berikut.

$$2n = \sum_{v \in V} \deg(v)$$

d. Lintasan (*Path*)
Dua buah simpul memiliki lintasan diantara keduanya jika terdapat kumpulan sisi-sisi yang jika ditelusuri dari simpul awal, dapat berakhir di simpul akhir.
e. Sirkuit (*Circuit*)
Sirkuit adalah sebuah lintasan yang berawal dan berakhir di simpul yang sama.
f. Terhubung (*Connected*)
Sebuah graf dikatakan sebagai graf terhubung jika untuk setiap pasang simpul dalam graf, terdapat minimal satu lintasan diantara kedua simpul tersebut. Jika ada satu saja pasangan simpul tidak memiliki lintasan diantaranya, maka graf tersebut bukan graf terhubung (graf tak-terhubung/*disconnected graph*)
g. Upagraf (*Subgraph*)
Sebuah upagraf dari sebuah graf adalah himpunan dari simpul-simpul tak kosong dan himpunan sisi-sisi, dimana himpunan simpul-simpulnya merupakan himpunan bagian dari himpunan simpul-simpul dari graf dan himpunan sisinya merupakan himpunan bagian dari himpunan sisi-sisi graf. Secara matematis, jika G = (V,E) dan $G_1 = (V_1,E_1)$ keduanya adalah graf, maka G_1 adalah upagraf dari G jika $V_1 \subseteq V$ dan $E_1 \subseteq E$[3]

A.3. Graf Lengkap (*Complete Graph*)

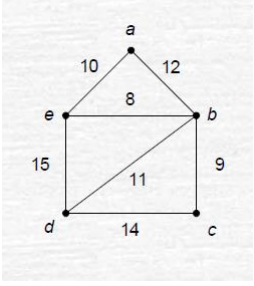
Salah satu graf sederhana yang akan sering digunakan dalam menyelesaikan persoalan *travelling salesman problem* adalah graf lengkap. Graf sederhana jenis ini memiliki simpul-simpul yang semua simpulnya berderajat *(n – 1)*, dimana *n* adalah jumlah simpul dalam graf. Jumlah sisi dalam graf ini adalah *n(n – 1)/2*.[4] Jika digambar, maka semua simpul dalam graf ini terhubung secara langsung, artinya untuk menuju dari suatu simpul ke simpul lain, pasti ada satu sisi yang bersisian dengan dua simpul tersebut.



Gambar 3. Contoh-contoh graf lengkap (Sumber : Rinaldi Munir, Matematika Diskrit rev.ed. 5, hlm. 377)

A.4. Graf Berbobot (*Weighted Graph*)

Sebuah graf yang sisinya diberikan suatu bobot disebut sebagai graf berbobot.[5] Bobot dalam graf jenis ini dapat merepresentasikan berbagai hal, tergantung graf yang ingin dibuat akan merepresentasikan apa. Dalam kasus *travelling salesman problem* yang akan dibahas dalam makalah ini, graf berbobot digunakan untuk merepresentasikan tempat-tempat, dengan simpul adalah tempat yang ingin dicapai, dan sisi adalah jarak antara tempat yang satu dengan tempat yang lain.



Gambar 4. Contoh graf berbobot. (Sumber : Rinaldi Munir, Matematika Diskrit rev.ed. 5, hlm. 376)

A.5. Lintasan Euler dan Sirkuit Euler

Lintasan Euler dari suatu graf adalah lintasan dalam graf yang memiliki semua sisi dari suatu graf tersebut. Sirkuit Euler dari suatu graf adalah sirkuit yang memiliki semua sisi dari suatu graf tersebut. Suatu graf yang memiliki sirkuit Euler pasti memiliki derajat setiap simpulnya genap, dan suatu graf yang memiliki lintasan Euler saja, namun bukan sirkuit Euler, memiliki tepat dua simpul berderajat genap. [6]

Konsep ini digunakan dalam pemecahan persoalan tukang pos Cina (*Chinese postman problem*), dimana terdapat seorang tukang pos yang akan mengantarkan surat ke beberapa tempat, namun harus melalui setiap jalan tepat sekali saja. Dalam algoritma *Christofides*, konsep sirkuit Euler juga digunakan dalam salah satu langkahnya.

A.6. Lintasan Hamilton dan Sirkuit Hamilton

Lintasan Hamilton dari suatu graf adalah lintasan dalam graf yang melewati semua simpul dari suatu graf tersebut tepat sekali. Sirkuit Hamilton dari suatu graf adalah sirkuit yang melewati semua simpul pada graf tepat sekali, kecuali simpul awal, yang juga merupakan simpul akhir.

Jika suatu graf sederhana dengan jumlah simpul $n \geq 3$ dan memiliki derajat setiap simpulnya $\geq n/2$, maka graf sederhana tersebut memiliki sirkuit Hamilton. Syarat adanya sirkuit Hamilton dari suatu graf diatas ini merupakan Teorema Dirac.[7] Akibat dari teorema ini, maka untuk setiap graf lengkap, pasti memiliki sirkuit Hamilton, karena untuk setiap graf lengkap, semua derajat simpulnya $n - 1$, yang pasti lebih besar dari $n/2$.

A.7. *Minimum-cost perfect matching*

Untuk sembarang graf tak-berarah berbobot G, sebuah *minimum-cost perfect matching* M adalah upagraf dari G yang memiliki semua simpul dari G, namun memiliki sisi-sisi sedemikian sehingga tidak ada sisi dalam himpunan sisi upagraf M yang bersisian dengan simpul yang sama dan memiliki bobot yang minimum. Dengan definisi seperti itu, maka syarat agar *minimum-cost perfect matching* dapat dicari adalah jumlah simpul dalam graf G harus genap.[8]

B. Pohon

Sebuah pohon adalah sebuah graf terhubung yang tidak memiliki sirkuit di dalam graf tersebut. Dengan definisi ini, maka sebuah pohon pastilah merupakan graf sederhana, karena pohon tidak mungkin mengandung kalang/*loop* maupun sisi ganda.[9]

B.1. Pohon Merentang Minimum

Sebuah pohon merentang minimum (*minimum spanning tree*) dari sebuah graf berbobot adalah pohon yang merupakan upagraf dari graf berbobot tersebut yang mengandung semua simpul dari graf berbobot tersebut dan jumlah bobot-bobot dalam upagraf tersebut haruslah yang paling kecil/minimum.[10]

Ada dua cara yang umum digunakan untuk mencari pohon merentang minimum dari suatu graf berbobot, yaitu dengan algoritma Prim dan algoritma Kruskal. Disini, akan dibahas cara mencari pohon merentang minimum menggunakan algoritma Prim saja.

Misalkan T adalah pohon yang akan dibentuk, G adalah graf berbobot, *n* adalah jumlah simpul dalam G, maka tahapan dalam algoritma Prim adalah sebagai berikut :

1. Ambil sisi dari G yang berbobot minimum, pindahkan ke T
2. Pilih sisi dari G dengan bobot terkecil, bersisian dengan simpul di T, dan tidak membentuk sirkuit di T. Masukkan sisi tersebut ke T
3. Ulangi langkah 2 sebanyak $n - 2$ kali.[11]

C. Kompleksitas Waktu Algoritma

Setiap algoritma pasti memerlukan waktu untuk memroses beberapa masukan dan mengeluarkan hasil yang diinginkan. Untuk menentukan berapa lama yang diperlukan oleh suatu algoritma, biasanya kita mengukur durasi eksekusi algoritma di suatu komputer dengan jumlah masukan yang berbeda-beda. Tetapi, sayangnya, cara ini bukanlah cara yang tepat untuk mengukur kebutuhan waktu suatu algoritma. Hal ini dikarenakan untuk arsitektur komputer yang berbeda dan untuk *compiler* yang berbeda akan menyebabkan waktu eksekusi algoritma yang berbeda-beda. Atas dasar itulah, dibuat sebuah model abstrak yang dapat menggambarkan waktu yang dibutuhkan oleh suatu algoritma untuk dieksekusi. Konsep inilah yang disebut sebagai kompleksitas waktu algoritma [12]

Dalam menentukan kompleksitas waktu sebuah algoritma, kita mencari berapa kali operasi dasar sebuah algoritma dijalankan. Operasi dasar adalah operasi yang menjadi ciri khas algoritma tersebut/inti tahapan dari algoritma tersebut. Kompleksitas waktu dilambangkan dengan $T(n)$.

Kompleksitas waktu sendiri masih dibedakan menjadi tiga jenis, yaitu $T_{max}(n)$, $T_{min}(n)$, dan $T_{avg}(n)$. $T_{max}(n)$ adalah waktu maksimal yang diperlukan oleh algoritma, $T_{min}(n)$ adalah waktu minimum yang diperlukan oleh suatu algoritma, dan $T_{avg}(n)$ adalah waktu rata-rata yang diperlukan oleh suatu algoritma.

Ada kalanya kita lebih membutuhkan cara untuk menggambarkan bagaimana kebutuhan waktu sebuah algoritma meningkat bersamaan dengan banyaknya jumlah masukan yang harus diproses. Hal tersebut kadang disebut sebagai kompleksitas waktu asimptotik. Hal tersebut dapat direpresentasikan dengan menggunakan notasi O-Besar (*Big-O*). Sebenarnya, ada dua lagi notasi yang sering digunakan, yaitu notasi Omega (Ω) dan notasi Tetha-Besar (Θ). Namun, notasi *Big-O* adalah notasi yang paling sering digunakan.

Suatu $T(n)$ adalah $O(f(n))$ ($T(n)$ berorde paling besar $f(n)$) jika ada suatu konstanta C dan n_0 sehingga $T(n) \leq C(f(n))$ terpenuhi untuk $n \geq n_0$.[13] Pernyataan tersebut adalah definisi dari notasi *Big-O*. Notasi ini menggambarkan bahwa waktu yang diperlukan oleh suatu algoritma paling besar akan berorde sama dengan $f(n)$.

III. PEMBAHASAN

A. Algoritma Christofides

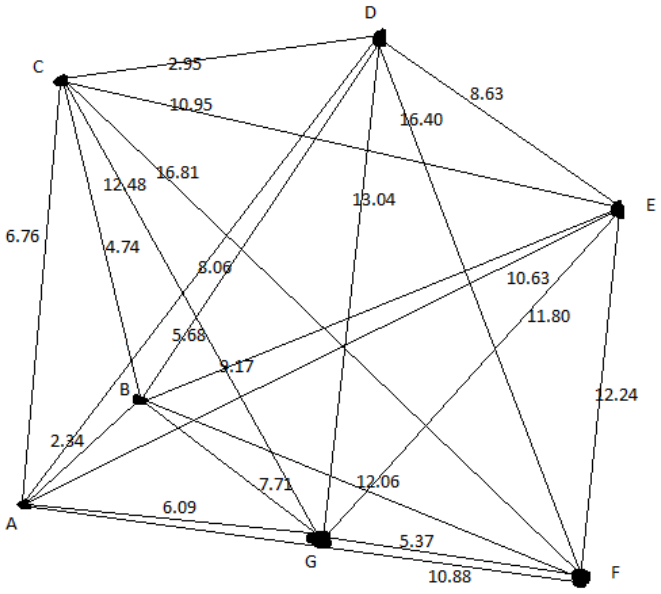
Algoritma *Christofides* adalah algoritma yang menghasilkan aproksimasi solusi dari *travelling salesman problem* dari sebuah graf tak berarah berbobot lengkap. Namun, ada satu syarat agar algoritma ini bekerja, yaitu graf masukan harus memenuhi

ketidaksamaan segitiga. Ketidaksamaan segitiga menyatakan bahwa untuk sembarang sisi dari segitiga, panjangnya harus lebih pendek dari jumlah panjang kedua sisi lainnya. Dalam hal graf berbobot, maka bobot suatu sisi dalam graf harus lebih kecil dari jumlah bobot dua buah sisi dalam graf, dimana simpul yang bersisian dengan sisi pertama sama dengan simpul awal dan simpul akhir yang dibentuk dari lintasan dua buah sisi.

Misalkan graf masukan adalah G. Secara umum, langkah-langkah untuk yang ditempuh dalam algoritma ini adalah sebagai berikut.

1. Bangun pohon merentang minimum, T, dari graf masukan G
2. Misalkan O adalah himpunan simpul dalam graf G berderajat ganjil. Menurut lemma jabat tangan, jumlah derajat suatu graf pastilah genap. Maka, banyaknya simpul dalam O akan selalu genap, agar jumlah semua derajat simpulnya genap. Jika H adalah graf dengan O sebagai simpulnya dan memiliki sisi-sisi dari graf G yang menghubungkan simpul-simpul dalam O, cari *minimum-cost perfect matching* M dari H.
3. Gabungkan graf T dan M, misal menjadi G', sehingga graf G' kemungkinan akan menjadi graf ganda (*multigraph*).
4. Cari sirkuit Euler dalam G'.
5. Ubah sirkuit Euler pada langkah ke 4 menjadi tur *travelling salesman problem*, dengan menghapus sisi-sisi yang membuat suatu simpul dikunjungi dua kali.[14]

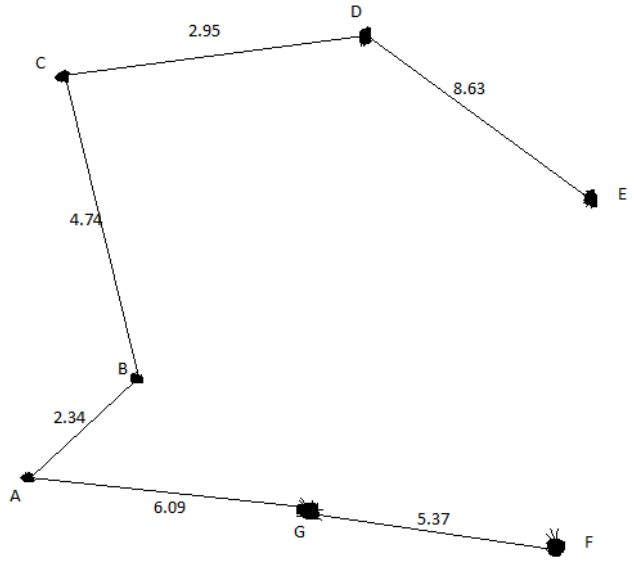
Untuk lebih jelasnya, perhatikanlah contoh graf lengkap dibawah ini.



Gambar 5. Sebuah graf lengkap berbobot

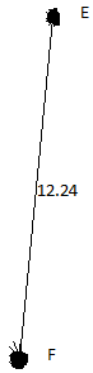
Dari graf pada Gambar 5, kita akan coba mencari jalur terpendek yang dimulai dari titik A, dengan melintasi setiap titik lainnya, dan kembali lagi ke titik A, atau dalam kata lain, kita akan mencari sirkuit Hamilton terpendek ditinjau dari titik A.

Dengan mengikuti langkah-langkah pada algoritma *Christofides*, maka pertama-tama, buatlah pohon merentang minimum dari graf diatas terlebih dahulu. Dengan algoritma Prim maupun Kruskal, akan didapatkan pohon merentang minimum seperti pada gambar 5.



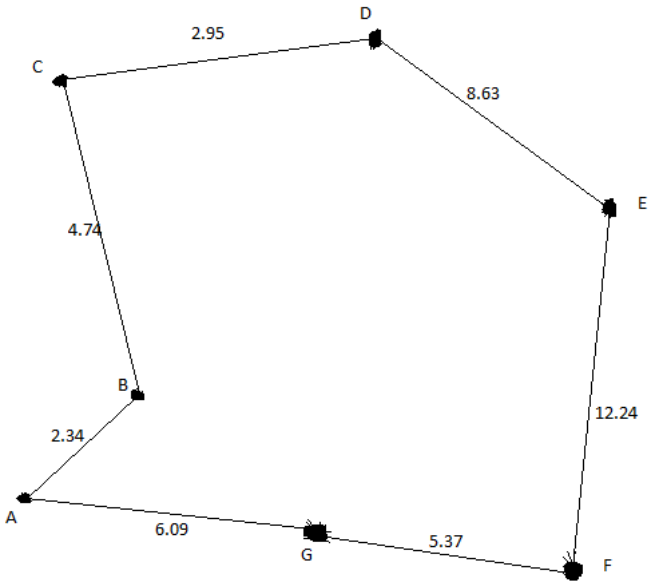
Gambar 6. Pohon merentang minimum dari graf pada Gambar 5.

Selanjutnya, dibuatlah himpunan simpul-simpul yang berderajat ganjil dari pohon merentang minimum pada Gambar 5. Hanya ada dua simpul yang berderajat ganjil pada pohon merentang minimum, yaitu simpul E dan simpul F. Dari himpunan simpul berrderajat ganjil itu, dibuatlah *minimum-cost perfect matching*. Karena hanya terdapat satu sisi yang bersisian dengan simpul E dan F, maka hasilnya sudah pasti sisi tersebut.



Gambar 7. Minimum-cost perfect matching dari simpul-simpul berderajat ganjil dari pohon merentang minimum pada Gambar 5.

Dari graf pada Gambar 5 dan Gambar 6, gabungkan kedua graf menjadi satu kesatuan graf. Sebagai pengingat, walaupun dalam contoh ini tidak ada sisi yang akan menjadi sisi ganda ketika graf disatukan, jika ditemukan hal seperti ini, jangan jadikan sisi ganda tersebut menjadi satu kesatuan sisi, biarkanlah menjadi sisi ganda. Sisi ganda tersebut nanti akan dieliminasi pada langkah ke-5.



Gambar 8. Gabungan graf pohon merentang minimum dengan graf minimum-cost perfect matching

Langkah selanjutnya adalah mencari sirkuit Euler dari graf gabungan pada Gambar 7. Dengan graf pada Gambar 7, sirkuit Euler yang dicari adalah (A,B),(B,C),(C,D),(D,E),(E,F),(F,G), dan (G,A).

Langkah terakhir adalah menghapus dan menambahkan sisi-sisi, sehingga tidak ada lagi simpul yang dikunjungi dua kali. Karena graf pada Gambar 7 juga sudah tidak lagi memiliki suatu simpul yang dikunjungi dua kali, maka graf pada Gambar 7 adalah sekaligus hasil akhir dari algoritma *Christofides* ini. Ditemukanlah perkiraan solusi persoalan pedagang keliling untuk graf pada Gambar 5, dengan total bobot = 2.34 + 4.74 + 2.95 + 8.63 + 12.24 + 5.37 + 6.09 = 42.36.

B. Perbandingan Algoritma Christofides dengan Algoritma Eksak Travelling Salesman Problem Lainnya

Untuk algoritma *Christofides* ini sendiri, operasi yang paling mendominasi dalam seluruh rangkaian algoritma adalah pada tahapan kedua, yaitu saat mencari *minimum-cost perfect matching*. Tahapan tersebut memiliki kompleksitas algoritma $O(n^3)$. Kemudian, hasil dari algoritma *Christofides* memiliki rentang keakuratan antara tepat/benar merupakan solusi eksak, sampai yang paling buruk adalah 1.5 kali solusi yang sebenarnya.[15]

Untuk algoritma yang mencari solusi *travelling salesman problem* secara tepat, ada dua cara yang cukup terkenal dalam mencari hal tersebut, yaitu dengan cara *brute force* dan dengan algoritma *Held-Karp*, yang merupakan aplikasi dari *dynamic programming*. Akan dibahas secara singkat kedua algoritma tersebut.

Dengan algoritma *brute force*, maka akan dicoba semua kemungkinan rute yang ada dalam graf yang ditinjau. Karena itu, maka jumlah perintah/iterasi yang dijalankan adalah sebanyak *n!* kali, dengan *n* adalah banyaknya tempat yang harus dikunjungi. Maka dari itu, algoritma *brute force* untuk menyelesaikan persoalan pedagang keliling memiliki kompleksitas waktu $O(n!)$.

Kemudian, dikembangkanlah cara lain yang menggunakan pendekatan *dynamic programming* oleh *Michael Held* dan *Richard Karp*. Pendekatan *dynamic programming* memecah persoalan yang ditinjau menjadi masalah-masalah yang lebih kecil, dan mencoba menyelesaikannya terlebih dahulu. Kemudian, untuk masalah yang lebih besar, digunakanlah hasil dari persoalan yang lebih kecil, sehingga mengurangi beban komputasi. Untuk algoritma ini sendiri, jumlah operasi yang dibutuhkan adalah $(n - 1)(n – 2)2^{n-3}$[16], sehingga algoritma ini akan memiliki kompleksitas $O(n^2 2^n)$.

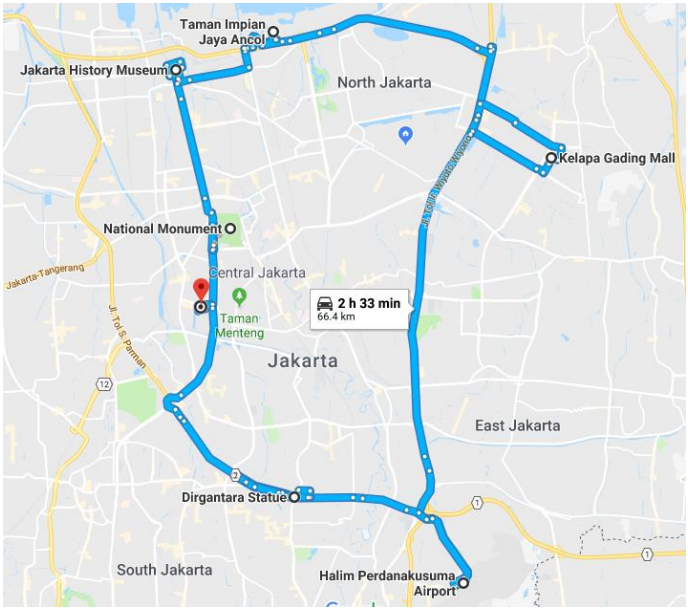
Maka dari itu, jika kedua cara tersebut dibandingkan dengan algoritma *Christofides*, secara kompleksitas waktu, algoritma *Christofides* jelas lebih baik dibandingkan dengan kedua algoritma yang mencari solusi *travelling salesman problem* secara eksak. Algoritma *Christofides* beroperasi dalam spektrum waktu polinomial, sedangkan dengan algoritma *brute force* dan *Held-Karp* beroperasi dalam spektrum waktu eksponensial. Sebagai perbandingan saja, jika merujuk pada contoh masalah pada bagian A, Gambar 5, dan berdasar pada kompleksitas waktu dari notasi *Big-O* tiap algoritma, jika terdapat $n = 7$ buah tempat untuk dianalisis, maka secara jumlah operasi, algoritma *Christofides* akan menjalankan 343 operasi, algoritma *brute force* akan menjalankan 5040 operasi, dan algoritma dengan *dynamic programming* akan menjalankan 6272 operasi. Dapat terlihat perbedaan jumlah operasi yang sangat signifikan antara ketiga algoritma tersebut.

Dengan melihat perbedaan waktu antara algoritma pencarian solusi *travelling salesman problem* ini, maka bisa dikatakan bahwa algoritma *Christofides* dapat dijadikan salah satu alternatif dalam mencari solusi dari *travelling salesman problem* ini. Walaupun hasil yang didapatkan masih dalam rentang galat yang cukup besar (50%/1.5 kali lebih panjang dibanding solusi eksak), algoritma *Christofides* memerlukan waktu yang jauh lebih sedikit dibandingkan algoritma yang eksak.

C. Aplikasi Travelling Salesman Problem

Sebagai pelengkap saja, akan dibahas sedikit mengenai persoalan dalam kehidupan sehari-hari yang dapat diselesaikan dengan memodelkan permasalahan tersebut dalam persoalan pedagang keliling ini.

Salah satu aplikasi yang dapat langsung diterjemahkan menjadi persoalan pedagang keliling adalah bagaimana merencanakan perjalanan wisata dalam suatu daerah. Biasanya, kalau sedang berlibur, kita akan mengunjungi beberapa tempat dari tempat dimana kita menginap, namun kembali lagi ke tempat menginap ketika semua tempat yang diinginkan sudah dikunjungi. Jika direpresentasikan dengan graf berbobot, maka tempat menginap dan tempat tujuan adalah simpul-simpul dalam graf, dan sisi-sisi dalam graf adalah jarak antar tempat-tempat tersebut.



Gambar 9. Ilustrasi rute suatu perjalanan wisata di kota Jakarta. (Sumber : maps.google.com)

Dalam aplikasi yang lebih nyata, seperti dalam *Google Maps*, representasi simpul-simpul dalam graf bukan hanya tempat-tempat tujuan saja, melainkan akan diperluas menjadi titik-titik persimpangan pada jalan. Sehingga, graf nantinya akan menjadi gambar/peta jalanan yang menyerupai aslinya, dengan sisi-sisi pada graf merepresentasikan jaringan jalan. Namun, dalam makalah ini, pada contoh di bab III bagian A, graf disederhanakan sehingga tidak melihat jalan sebenarnya, hanya mengambil jarak terpendek antar dua titik saja. (contoh pada bagian tersebut sebenarnya juga merupakan representasi sederhana dari peta pada Gambar 9.)

Aplikasi lainnya yang tidak akan dibahas dalam makalah ini adalah dalam proses pembuatan *microchip*, perencanaan, perancangan logistik, dan bila dimodifikasi, bisa bermanfaat dalam pembentukan urutan suatu DNA.

IV. SIMPULAN

Dari penjabaran pada bagian II dan III dari makalah ini, maka dapat disimpulkan bahwa algoritma *Christofides* dapat menaksir solusi dari persoalan pedagang keliling. Hasil perkiraan dari algoritma *Christofides* memiliki rentang galat maksimal 50%. Dibandingkan dengan algoritma tradisional untuk mencari solusi eksak dari *travelling salesman problem*, algoritma *Christofides* memerlukan waktu yang lebih singkat, dengan kompleksitas waktu $O(n^3)$. Dengan kompleksitas waktu yang jauh lebih singkat serta rentang galat yang terbentuk, algoritma *Christofides* ini dapat menjadi alternatif cara untuk diterapkan dalam aplikasi yang menggunakan konsep *travelling salesman problem*, seperti dalam menentukan rute yang mangkus dalam sebuah wisata/kunjungan, merencanakan jaringan logistik, bahkan, jika diubah sedikit, bisa membantu dalam proses penentuan urutan DNA (*DNA sequencing*).

V. UCAPAN TERIMA KASIH

Pertama-tama, saya ingin mengucapkan syukur kepada Tuhan YME, karena dengan bimbingan dan rahmat-Nya, saya bisa menyelesaikan makalah ini.

Kemudian, saya juga ingin mengucapkan terima kasih kepada teman-teman yang sudah memberikan saya inspirasi, masukan dan dukungan selama saya membuat makalah ini. Terakhir, saya ingin mengucapkan terima kasih kepada orangtua saya, yang telah memberikan dukungan untuk saya.

DAFTAR PUSTAKA

[1] K.H. Rosen, *Discrete Mathematics and Its Applications*, rev. ed. 7, New York: McGraw-Hill, 2012, pp. 641.
[2] Rinaldi Munir, *Matematika Diskrit, rev. ed. 5*,Bandung: Informatika, 2014,pp. 357 - 359.
[3] *Ibid*., pp. 364 - 373
[4] *Ibid*., pp. 377
[5] *Ibid*., pp. 376
[6] K.H. Rosen, *op.cit.*, pp. 693 - 697
[7] *Ibid*., pp. 698 – 701
[8] Michael T. Goodrich, *Algorithm Design & Applications*, New Jersey: John Wiley & Sons, 2015, pp. 513
[9] K.H. Rosen, *op. cit.*, pp. 746.
[10] *Ibid*., pp. 785,798
[11] Rinaldi Munir,*op.cit.*, pp. 450 – 451
[12] *Ibid*., pp. 498 – 499
[13] *Ibid*., pp. 511
[14] Michael T. Goodrich, *op.cit.*
[15] *Ibid*., pp. 514
[16] Michael Held & Richard M. Karp, *A Dynamic Programming Approach to Sequencing Problems*, https://ucilnica1213.fmf.uni-lj.si/pluginfile.php/11706/mod_resource/content/0/HELDKarpAlgoritemZaPTP_clanek.pdf , terakhir diakses 3 Desember 2017, 14.17

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 3 Desember 2017

Antonio Setya
13516002