**Hak cipta dan penggunaan kembali:**

Lisensi ini mengizinkan setiap orang untuk menggubah, memperbaiki, dan membuat ciptaan turunan bukan untuk kepentingan komersial, selama anda mencantumkan nama penulis dan melisensikan ciptaan turunan dengan syarat yang serupa dengan ciptaan asli.

**Copyright and reuse:**

This license lets you remix, tweak, and build upon work non-commercially, as long as you credit the origin creator and license it on your new creations under the identical terms.

Team project ©2017
Dony Pratidana S. Hum | Bima Agus Setyawan S. IIP

# BAB II

# TELAAH LITERATUR

## 2.1 Pengertian dan Fungsi Gudang

Menurut kamus besar bahasa Indonesia (KBBI), pengertian gudang yaitu rumah atau bangsal tempat menyimpan barang-barang. Menurut (Munif, 2008) Gudang dapat didefinisikan sebagai tempat yang dibebani tugas untuk menyimpan barang yang akan dipergunakan dalam produksi sampai barang diminta sesuai dengan jadwal produksi. Sejak dulu, gudang berfungsi sebagai *buffer* atau penyeimbang dan untuk menentukan langkah selanjutnya suatu perusahaan, apakah perusahaan akan menggunakan gudang untuk komersial atau lebih baik digunakan sendiri.

Gudang sebagai tempat penyimpanan produk atau barang untuk memenuhi permintaan pelanggan secara cepat dan mempunyai beberapa fungsi diantara penerimaan dan pengiriman produk. Fungsi-fungsi pokok gudang sebagai berikut:

1. *Receiving* (penerimaan) dan *shipping* (pengiriman).
2. *Identifying and shorting* (pengidentifikasian dan penyaringan).
3. *Dispatching* ke penyimpanan.
4. *Picking the order* (pemilihan pesanan).
5. *Storing* (penyimpanan).
6. *Assembling the order* (perakitan pesanan).
7. *Packaging* (pengepakan).



Aplikasi Simulasi ..., Maulana Sani, FTI UMN, 2014

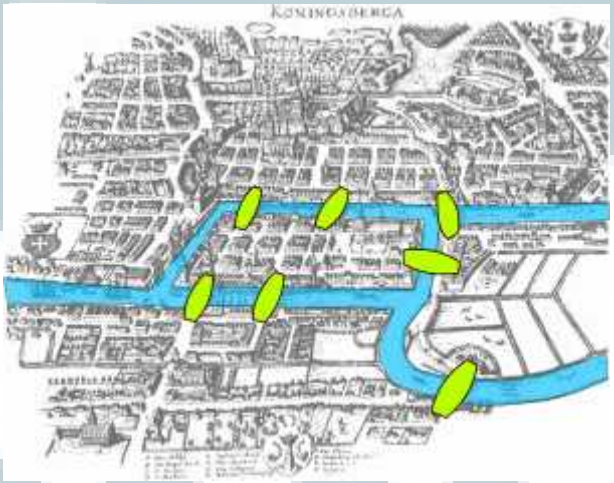
8. *Dispatching the shipment* (pengiriman).

9. *Maintaining record* (perawatan produk).

## 2.2 Teori Graph

### 2.2.1 Pengenalan Teori *Graph*

Teori *graph* adalah cabang ilmu yang mempelajari sifat-sifat *graph* yang pertama kali diperkenalkan pada tahun 1736. Baru pada sekitar tahun 1920 teori *graph* berkembang pesat terutama di bidang ilmu komputer, kimia, bahasa, ekonomi, dan riset operasi.

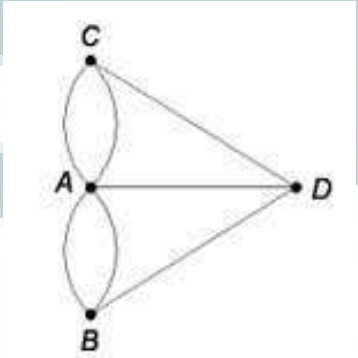


Gambar 2.1. Tujuh jembatan utama di *Königsberg*
(Sumber: Munir, 2005, p.355)

Menurut catatan sejarah, masalah jembatan *Königsberg* adalah masalah yang pertama kali menggunakan *graph* pada tahun 1739 (Rinaldi, 2005, p.355). Di kota *Königsberg* (sebelah timur negara bagian *Prussia*, Jerman) dan sekarang bernama kota *Kaliningrad*, terdapat sungai *Pregal* yang mengalir mengitari Pulau *Kneiphof* lalu bercabang menjadi dua buah anak sungai. Ada tujuh buah jembatan yang

menghubungkan daratan yang dibelah oleh sungai tersebut seperti terlihat dalam gambar 2.1. Permasalahannya adalah menemukan pejalanan atau rute dari suatu kota melalui ketujuh buah jembatan masing-masing tepat satu kali, kemudian kembali lagi ke tempat awal. Pulau tersebut tidak dapat dicapai oleh rute apapun selain melalui jembatan-jembatan tersebut.

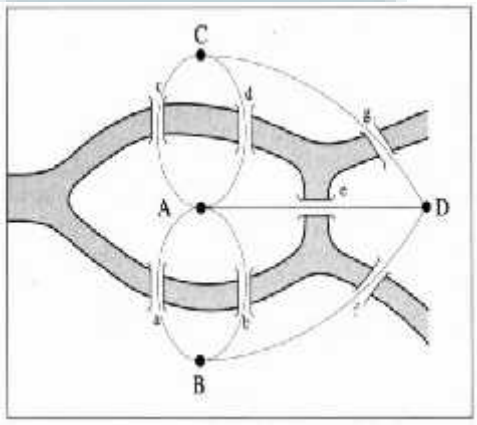
Tahun 1736, *Leonhard Euler* adalah orang pertama yang berhasil menemukan jawaban masalah tersebut dengan pembuktian yang sederhana (melalui karya tulisannya *Seven Bridges of Königsberg*). Daratan (titik-titik yang dihubungkan oleh jembatan) dinyatakan sebagai titik disebut verteks dan jembatan dinyatakan sebagai edge. Dari analisa *Euler* pada jembatan *Königsberg* menghasilkan sebuah model *graph*, seperti yang diperlihatkan pada gambar 2.2.




Gambar 2.2. *Graph* yang merepresentasikan jembatan *Königsberg*
(Sumber: Munir, 2005, p.355)

Analisis *Euler* mengenai permasalahan jembatan di *Königsberg* tidak menghasilkan solusi. Karena orang tidak mungkin melalui ketujuh jembatan masing-masing tepat satu kali dan kembali lagi ke tempat awal keberangkatan jika derajat (banyaknya garis yang bersisian dengan titik) setiap verteks tidak seluruhnya genap. Penemuan *Euler* adalah kunci yang menandai perkembangan

topologi, di mana perbedaan antara *layout* sebenarnya dan *scematic graph* adalah contoh yang bagus untuk gagasan bahwa topologi tidak dibatasi dengan bentuk kaku dari objek-objek tertentu.




Gambar 2.3. Gabungan gambar jembatan *Königsberg* dan *graph*
(Sumber: *Barabasi*, 2009, p.11)

### 2.2.2 Definisi *Graph*

*Graph* adalah kumpulan verteks atau *node* yang dihubungkan satu sama lain melalui sisi/rusuk/busur/*edge*, yang digunakan untuk merepresentasikan objek-objek diskrit dan hubungan antara objek-objek tersebut. Menurut (Rosen, 2007, p.589) suatu *graph* (*undirected graph*) yang bila dinotasikan adalah G = (V, E). *Graph G* didefinisikan sebagai pasangan himpunan (V,E), ditulis dengan notasi *G(V,E)*, yang dalam hal ini.

i. *V* adalah himpunan tidak kosong dari simpul-simpul (titik/verteks/*node*).
ii. *E* adalah himpunan sisi (rusuk/*edge*) yan menghubungkan sepasang simpul.

Verteks-verteks pada *graph* dapat merupakan obyek sembarang seperti kota, atom-atom suatu zat, nama anak, jenis buah, komponen alat elektronik dan

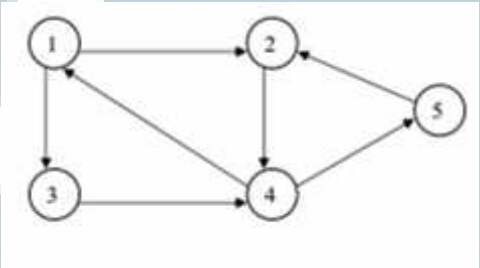
sebagainya. *Edge* dapat menunjukkan hubungan sembarang seperti rute penerbangan, jalan raya, sambungan telepon, ikatan kimia, dan lain-lain. Jika terdapat sebuah rusuk *e* yang menghubungkan verteks *v* dan *w*, ditulis edge (*v*, *w*).

### 2.2.3 Jenis Graph

Menurut *Johnsonbaugh*, *graph* dapat dibagi menjadi 2 jenis berdasarkan arahnya yaitu sebagai berikut (Johnsonbaugh, 2009).

1. *Graph* berarah (*directed graph*)

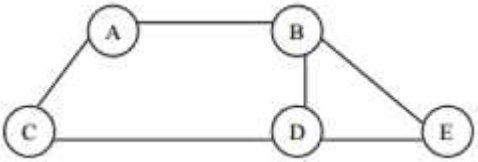
   *Graph* yang setiap sisinya diberikan orientasi arah, *Edge* (*v*,*w*) ≠ *edge* (*w*,*v*)




Gambar 2.4. Contoh gambar *graph* berarah
(Sumber: *Johnsonbaugh*, 2009, p.378)

2. *Graph* tidak berarah (*undirected graph*)

   *Graph* yang sisinya tidak mempunyai orientasi arah. *Edge* (v, w) = *edge* (w, v) adalah sisi yang sama

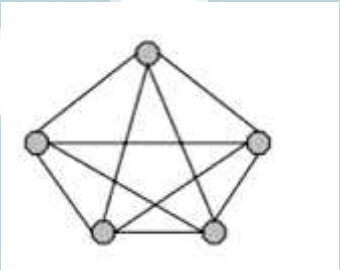



Gambar 2.5. Contoh gambar *graph* tidak berarah
(Sumber: *Johnsonbaugh*, 2009, p.378)

Sebuah struktur *graph* bisa dikembangkan dengan memberi bobot atau nilai pada tiap *edge* di mana merupakan suatu nilai yang dapat berupa biaya atau jarak,

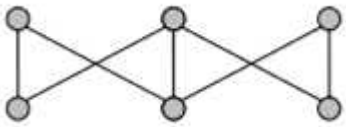
*graph* semacam ini disebut *graph* berbobot (*weighted graph*). Dalam pengajaran teori *graph* (Lipschutz, 1985, p.85), terdapat *graph* khusus beberapa diantaranya adalah sebagai berikut.

a. *Complete graph* ialah graph di mana setiap verteks berhubungan dengan semua verteks yang lain (semua *vertex* saling berhubungan). Biasanya direpresentasikan dengan simbol $K_n$, dimana K adalah *complete graph* dan *n* jumlah verteks. Sebuah *complete graph* dengan *n* verteks akan mempunyai rusuk sebanyak $n(n-1)/2$.



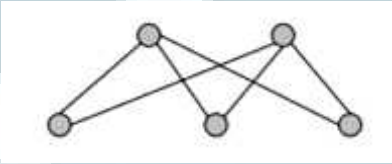
Gambar 2.6. Contoh model *complete graph* ($K_5$)
(Sumber: *Lipschutz*, 1985, p.85)

b. *Bipartite graph* adalah graph dimana satu verteksnya dibagi ke dalam dua subset verteks *m* dan *n*, sedemikian sehingga tidak ada rusuk yang menyebabkan verteks-verteks dalam *subset* yang sama. Biasanya direpresentasikan dengan simbol $K_{m,n}$ , di mana K adalah *bipartite graph*, dan m adalah jumlah *sunset* verteks *m*, dan *n* adalah jumlah *subset n*.



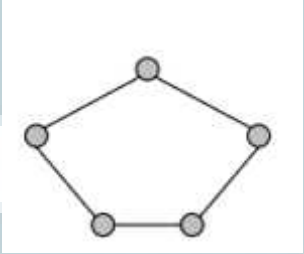
Gambar 2.7. Contoh model *bipartite graph* ($K_{3,3}$)
(Sumber: *Lipschutz*, 1985, p.86)

c. *Complete bipartite graph* adalah *bipartite graph* dimana setiap verteks dari *m* harus memiliki rusuk yang berhubungan ke semua verteks dari *n*. Biasanya direpresentasikan dengan simbol $K_{m,n}$ , sama seperti *bipartite graph*.



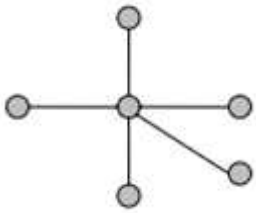
Gambar 2.8. Contoh model *bipartite graph* ($K_{2,3}$)
(Sumber: *Lipschutz*, 1985, p.86)

d. *Regular graph* adalah *graph* dimana setiap verteksnya memiliki derajat yang sama.



Gambar 2.9. Contoh model *regular graph* berderajat 2
(Sumber: *Lipschutz*, 1985, p.85)

e. *Tree* adalah *graph* yang tidak memiliki *cycle*. Jika jumlah verteks pada tree adalah n, maka jumlah rusuk pada tree adalah n-1.



Gambar 2.10. Contoh model *tree graph*
(Sumber: *Lipschutz*, 1985, p.86)

### 2.2.4 Teori Lintasan dan Siklus

Misalkan $v_o$ dan $v_n$ adalah verteks-verteks dalam sebuah *graph* (Johnsonbaugh, 2002, p.12) Sebuah lintasan dari $v_o$ ke $v_n$ dengan panjang *n* adalah sebuah barisan berselang-seling dari (*n+1*) verteks dan *n edge* yang berawal dengan verteks $v_0$ dan berakhir dengan verteks *vn*,

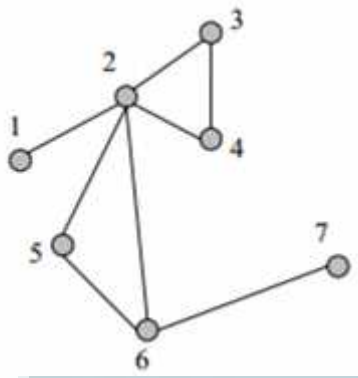
$$(v_0, e_1, v_1, e_2, v_2, \ldots, v_{n-1}, e_n, v_n)$$

Dengan rusuk $e_i$ insiden pada verteks $v_{i-1}$ dan $v_i$ ( $i$ = 1, 2, …, $n$).

Sebuah siklus adalah sebuah lintasan yang mempunyai panjang lintasan tidak nol dari kota pertama sampai kota terakhir yang merupakan kota pertama, dimana tidak terdapat rusuk yang dilalui lebih dari sekali.

Sebuah siklus sederhana adalah siklus dari kota pertama sampai kota terakhir yang merupakan kota terakhir juga pada suatu *graph*, yang kecuali kota pertama dan kota terakhir yang sama, tidak terdapat verteks yang berulang.

Untuk mengamati perbedaan antara lintasa, siklus, siklus sederhana, dengan contoh *graph* pada gambar 2.11 dapat dilihat dalam bentuk tabel.




Gambar 2.11. *Graph* tidak berarah

(Sumber: Johnsonbaugh, 2002, p.12)

Tabel 2.1. Perbedaan Lintasan, Siklus dan Siklus Sederhana

(Sumber: Johnsonbaugh, 2002, p.12)

| **Lintasan** | **Lintasan Sederhana** | **Siklus** | **Siklus Sederhana** |
|---|---|---|---|
| (5, 6, 2, 5) | Tidak | Ya | Ya |
| (2, 6, 5, 2, 4, 3, 2) | Tidak | Ya | Tidak |
| (6, 5, 2, 4) | Ya | Tidak | Tidak |
| (6, 5, 2, 4, 3, 2, 1) | Tidak | Tidak | Tidak |

### 2.2.5 Representasi *Graph*

Misalkan *G* adalah sebuah *graph* dengan verteks-verteks $v_1, v_2, \ldots, v_n$ dan *edge* $e_1, e_2, \ldots, e_n$ maka didefinisikan dua macam matriks yang berhubungan dengan sebuah *graph* G (Lipschutz, 1985, p.87)

1. Matriks *adjacency*

Misalkan $A=(a_{ij})$ adalah matriks $m$ x $n$ yang didefinisikan oleh:

$$a_{ij} = \begin{cases} 1 & \text{jika } \{u, v\} \text{ adalah } edge, \text{ yaitu } v_i \text{ } adjacent \text{ terhadap } v_j \\ 0 & \text{lainnya} \end{cases}$$

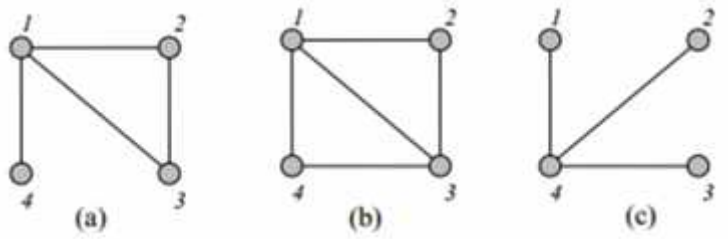
2. Matriks *incident*

Misalkan M=(mij) adalah matriks m x i didefinisikan oleh:

$$m = \begin{cases} 1 & \text{verteks } v_i \text{ adalah } incident \text{ pada } edge\ e_i \\ 0 & lainnya \end{cases}$$

### 2.2.6 *Graph* Hamilton

Lintasan Hamilton ialah lintasan yang melalui tiap verteks di dalam *graph* tepat satu kali (Munir, 2005, p.408). Sirkuit Hamilton ialah sirkuit yang melalui tiap verteks di dalam *graph* tepat satu kali, kecuali verteks asal (sekaligus verteks akhir) yang dilalui dua kali. *Graph* yang memiliki sirkuit Hamilton dinamakan *graph* Hamilton, sedangkan *graph* yang hanya memiliki lintasan Hamilton disebut *graph* semi-Hamilton.




Gambar 2.12. Penggambaran *Graph* Hamilton

(Sumber: Munir, 2005, p.409)

## 2.3 Algoritma

Berdasarkan buku Computer Algorithms/C++ (Horowitz, et al., 1998, p.1) Algoritma berasal dari kata Algoris dan Ritmis, yang pertama kali diungkapkan oleh Abu Ja'far Mohammad Ibn Musa Al Khowarizmi dalam buku *Al-jabrwa'almuqabala* (*Rules of Restoration and Reduction*).

Ada beberapa definisi yang diberikan mengenai pengertian algoritma antara lain sebagai berikut:

1. Menurut Abu Ja'far Mohammad Ibn Musa Al Khowarizmi Algoritma adalah suatu metode khusus untuk menyelesaikan suatu permasalahan.
2. Menurut Goodman Hedetniemi Algoritma adalah urut-urutan terbatas dari operasi-operasi yang terdefinisi dengan baik, yang masing-masing membutuhkan memori dan waktu yang terbatas untuk menyelesaikan suatu permasalahan.
3. Dalam ilmu komputer Algoritma adalah suatu metode yang terdiri dari serangkaian langkah-langkah yang terstruktur dan dituliskan secara sistematis yang akan dikerjakan untuk menyelesaikan masalah dengan bantuan komputer.

Secara umum algoritma ialah sejumlah langkah komputasi yang mengubah masukan (input) menjadi keluaran (output) yang benar (Prabowo, 2009). Menurut *Donald E. Knuth* sebuah algoritma harus memenuhi persyaratan:

- *Finiteness*: algoritma harus berakhir setelah melakukan sejumlah langkah proses.

- *Definiteness*: setiap langkah algoritma harus didefinisikan dengan tepat dan tidak menimbulkan makna ganda (*ambiguous*).
- *Input*: setiap algoritma memerlukan data sebagai masukan untuk diolah.
- *Output*: setiap algoritma memberikan satu atau beberapa hasil keluaran.
- *Effectiveness*: langkah-langkah algoritma dikerjakan dalam waktu yang wajar.

### 2.3.1 Algoritma Floyd-Warshall

Algoritma *Floyd-Warshall* merupakan salah satu varian dari pemrograman dinamis, yaitu suatu metode yang melakukan pemecahan masalah dengan memandang solusi yang akan diperoleh sebagai suatu keputusan yang saling terkait dan lebih menjamin keberhasilan penemuan solusi optimum untuk kasus penentuan lintasan terpendek (*all pair shortest path*), Artinya solusi-solusi tersebut dibentuk dari solusi yang berasal dari tahap sebelumnya dan ada kemungkinan solusi lebih dari satu (Novandi, 2007, p.2).

$f_k\,(s) = cx_k s$ .... Rumus 2.1. Perhitungan tahapan solusi

Rekurens:

$$f_k(s) = min\, x_k\,\{cx_k s + f_k - 1(x_k)\}, k = 2, 3, 4.$$

Algoritma *Floyd-Warshall* memiliki input *graph* berarah dan berbobot (V,E), yang berupa daftar titik (*node/vertex* V) dan daftar sisi (*edge* E). Jumlah bobot sisi-sisi pada sebuah jalur adalah bobot jalur tersebut (Cormen, et al., 2009).

Dalam mendesain denah gudang, tiap-tiap *aisle* (lorong) diberi titik awal dan titik akhir, ini digunakan untuk memperoleh daftar titik (*node/vertex*). Sisi (*edge* E) adalah *aisle*. Dan tiap ruas sisi/*edge* mempunyai bobot yaitu waktu tempuh dalam satuan detik. Sisi pada E diperbolehkan memiliki bobot negatif, akan tetapi tidak diperbolehkan bagi *graph* untuk aplikasi ini untuk memiliki bobot negatif (Cormen, et al., 2009).

Algoritma ini menghitung bobot terkecil dari semua jalur yang menghubungkan sebuah pasangan titik, dan melakukannya sekaligus untuk semua pasangan titik. Dengan kata lain pada saat perhitungan rute optimum yang akan dilalui terlebih dahulu menghitung semua kemungkinan rute yang akan dilalui kemudian baru mencari rute optimum dengan cara membandingkan tiap pasangan rute apakah ada pasangan rute lain yang lebih optimum.

### 2.3.2 Algoritma *2-opt*

Algoritma *2-opt* yang merupakan algoritma heuristik yang merupakan algortima yang mencari nilai optimal dari bagian atau irisan suatu permasalahan utama lalu melakukan proses penyempurnaan sehingga mencapai solusi tebaik yang diinginkan dan digunakan untuk mencari solusi terbaik dengan cara memperbaiki solusi yang ada. Cara kerja algoritma ini sama dengan metode mutasi *swap* pada algoritma genetik, yaitu dengan memilih dua elemen untuk kemudian menukar posisinya. Algoritma *2-opt* tidak mencari seluruh permutasi dari sebuah solusi melainkan hanya sebagian kecil saja, karena elemen yang diubah hanyalah dua elemen. Algoritma ini memiliki kompleksitas $O(n^2)$. Hal ini

didapatkan berdasarkan cara kerja algoritma ini yang melakukan perhitungan seperti dibawah ini (Hasegawa, et al., 1997 pp. 2344-2347).

$C\ (n, 2) = \frac{n!}{(2! * (n-2)!)}$ .... Rumus 2.2. Perhitungan cara kerja algoritma *2-Opt*

(Sumber: Hasegawa, et al., 1997 p. 2347)

## 2.4 Optimasi

### 2.4.1 Definisi Optimasi

Optimisasi ialah suatu proses untuk mencapai hasil yang ideal atau optimal (nilai efektif yang dapat dicapai) (Prabowo, 2009). Dalam matematika, optimisasi merujuk pada studi permasalahan yang mencoba untuk mencari nilai minimal atau maksimal dari suatu fungsi riil. Untuk dapat mencapai nilai optimal baik minimal atau maksimal tersebut, secara sistematis dilakukan pemilihan nilai variabel integer atau riil yang akan memberikan solusi optimal. Nilai optimal adalah nilai yang didapat melalui suatu proses dan dianggap menjadi suatu solusi jawaban yang paling baik dari semua solusi yang ada.

### 2.4.2 Macam-Macam Permasalahan Optimisasi

Persoalan yang berkaitan dengan optimisasi sangat kompleks dalam kehidupan sehari-hari. Nilai optimal yang didapat dalam optimisasi dapat berupa besaran panjang, waktu, jarak, dan lain-lain. Berikut ini adalah beberapa persoalan yang memerlukan optimisasi (Prabowo, 2009):

a. Menentukan lintasan terpendek dari suatu tempat ke tempat yang lain.

b. Menentukan jumlah pekerja seminimal mungkin untuk melakukan suatu proses produksi agar pengeluaran biaya pekerja dapat diminimalkan dan hasil produksi tetap maksimal.
c. Mengatur jalur kendaraan umum agar semua lokasi dapat dijangkau.
d. Mengatur *routing* jaringan kabel telepon agar biaya pemasangan kabel tidak terlalu besar dan penggunaannya tidak boros.

Selain beberapa contoh di atas, masih banyak persoalan lainnya yang terdapat dalam kehidupan sehari-hari.

## 2.5 Travelling Salesman Problem

*Travelling Salesman Problem* (TSP) adalah permasalahan pada metematika diskrit dan optimalisasi kombinatorial yang berhubungan dengan permasalahan dalam menemukan satu siklus hamiltonian dalam suatu grafik. *Travelling Salesman Problem* (*TSP*) pertama kali dikemukakan pada tahun 1800-an oleh seorang matematikawan asal Irlandia, *Sir William Hamilton* dan seorang matematikawan asal Inggris, *Thomas Penyngton Kirkman*. Bentuk umum dari *Travelling Salesman Problem* (*TSP*) pertama dipelajari oleh para matematikawan mulai tahun 1930. Diawali oleh *Karl Menger* di *Vienna* dan *Harvard*. Setelah itu permasalahan *Travelling Salesman Problem* (*TSP*) dipublikasikan oleh *Hasler Whitney* dan *Merrill Flood* di *Princeton*. Penelitian secara rinci mengenai hubungan anatara penemuan *Menger* dan *Whitney*, serta perkembangan *Travelling Salesman Problem* (*TSP*) dapat ditemukan dalam makalah *Alexander Schrijver "On the history of combinatorial optimization (till 1960)"*.

*Travelling Salesman Problem* (*TSP*) memerlukan keputusan dari siklus biaya minimal, yang melalui setiap simpul pada *graph* yang relevan. Jika biaya bersifat simetris, yakni jika biaya antar dua lokasi tidak tergantung pada arah lintasan, maka *Travelling Salesman Problem* (*TSP*) jenis ini bersifat simetris, sedangkan dalam hal lain bersifat asimetris, yang disebut *Travelling Salesman Problem* (*TSP*) berarah.

Solusi-solusi dari berbagai metode dianalisis dan setiap *n* buah simpul memerlukan sejumlah waktu *f(n)*. Besaran *f(n)* fungsi waktu dari sebuah metode terhadap jumlah kota *n*. Untuk membandingkan dua buah metode, cukup dibandingkan fungsi *f(n)* dari masing-masing metode. Suatu metode mungkin baik tetapi analisisnya buruk, sehingga menghasilkan *f(n)* yang buruk. Dalam banyak permasalahan komputasi, studi tentang algoritma dan fungsi telah menghasilkan solusi matematika yang sangat bagus, yang penting dalam penyelesaian permasalahan praktis. Hal ini telah menjadi subjek studi yang utama dalam sains komputer.

Untuk setiap masalah *Travelling Salesman Problem* (*TSP*) fungsi *f(n)* adalah *(n-1)! = (n-1) (n-2) (n-3)*...(3),(2),(1) dan jumlah jalur yang mungkin tejadi adalah *(n-1)!/2*. Hasil yang lebih baik ditemukan pada tahun 1962 oleh *Michael Held* dan *Richard Karp*, yang menemukan algoritma dan fungsi yang mempunyai $f(n) = n^2\ 2^n$, yaitu n x n x 2 x 2 x … x 2. Untuk setiap n bernilai besar, fungsi *f(n) Held-Karp* akan selalu lebih kecil dibandingkan *(n-1)!*.