Perubahan Persepsi pada Algoritma Dijkstra

Ferry Pangaribuan – NIM : 13505080

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15080@students.if.itb.ac.id

Abstrak
Algoritma Dijkstra salah satu algoritma paling populer di ilmu komputer. Ini juga popular pada operasi pencarian. Ini mendapat perhatian dari dosen-dosen dan mahasiswa. Sayangnya, jumlah fitur-fitur penting dari algoritma yang mempesonakan ini tidak transparan kepada pemakai rata-rata sebagaimana seharusnya. Pada umumnya, satu konsekuensi dari sejarah dari algoritma tersebut adalah algortima tersebut dipandang sebagai metode ilmu komputer daripada sebagai metode operasi pencarian.

Pada makalah ini mencoba menukar persepsi tersebut dengan menyediakan sebuah perspektif OR/MS perspective pada algoritma tersebut. Pada umumnya, kita mengingat bahwa algoritma terkenal ini diinspirasi dengan sangat kuat oleh *Bellman's Principle of Optimality* dan keduanya secara konseptual dan secara teknikal ini merupakan sebuah pemrograman dinamik *successive approximation*. Satu dari aplikasi yang dekat dari perspektif ini bahwa algoritma popular ini dapat tergabung pada silabus pemrograman dinamik dan pada gilirannya harus disinggung pada sebuah penjelasan yang terperinci dari algoritma tersebut.

1. Pendahuluan

Tahun 1959 sebuah tulisan sepanjang tiga halaman yang berjudul *A Note on Two Problems in Connexion with Graphs* diterbitkan pada jurnal *Numerische Mathematik.* Pada tulisan ini, Edsger W. Dijkstra – seorang ilmuwan computer berumur dua puluh sembilan tahun - mengusulkan algoritma-algoritma untuk solusi dari dua masalah teoritis graf dasar: the *minimum weight spanning tree problem* and the *shortest path problem.* Pada makalah ini akan diulas sedikit mengenai tilisan tersebut. Algoritma Dijkstra untuk masalah jalan terpendek adalah satu dari algoritma-algoritma palaing ternama pada ilmu komputer dan sebuah algoritma paling popular pada oparasi pencarian(OR).

Pada literatur tersebut, algoritma ini sering digambarkan sebagai sebuah algoritma yang tamak. Contohnya, buku *Algorithmics* (Brassard and Bratley [1988, pp. 87-92]) mengulas ini pada bab tersebut dengan judul *Greedy Algorithms. Encyclopedia of Operations Research and Management Science* (Gass and Harris [1996, pp. 166-167]) menggambarkan algoritma ini sebagai sebuah "*node labelling greedy algorithm* " dan sebuah algoritma yang tamak digambarkan sebagai "*a heuristic algorithm that at every step selects the best choice available at that step without regard to future consequences* " (Gass and Harris [1996, p. 264]).

Walaupun algoritma tersebut sangat terkenal pada literature OR/MS tersebut, ini secara umum dipandang sebagai sebuah ”metode ilmu komputer”. Rupanya ini karena tiga faktor: (a) penemunya adalah seorang ilmuwan komputer (b) hubungannya dengan struktur-struktur data khusus, dan (c) ada persaingan algoritma-algoritma berorientasi OR/MS untuk masalh jalan terpendek. Ini tidak mengejutkan karena beberapa disusun dengan baik *textbooks* OR/MS bahkan tidak menyebutkan algoritma ini dalam diskusi mereka pada masalah jalan terpendek (contoh. Daellenbach et al [1983], Hillier and Lieberman [1990]) dan semua yang dibicarakan ini pada konteks sekarang ini pada sebuah cara yang berdiri sendiri, bahwa, mereka tidak mengkaitkan ini ke metode standar OR/MS (contoh. Markland and Sweigart [1987], Winston [2004]). Untuk alas an yang sama, ini tidak mendejutkan bahea "Algoritma Dijkstra" terdapat pada *Encyclopedia of Operations Research and Management Science* (Gass and Harris [1996, pp. 166-167]) tidak memeliki referensi apapun untuk pemrograman dinamik.

Fokus dari makalah ini adalah untuk memperkenalakan cara membawakan yang sama sekali berbeda dari Algoritma Dijkstra, originalitasnya, perannya dalam OR/MS dan CS, dan hubungannya ke metode-metode dan teknik-teknik OR/MS yang lain. Sebagai pengganti dari pemusatan pada kompleksitas perhitungan, struktru-struktur data bahwa ini

dapat memanfaatkan untuk pengaruh baik, dan persoalan implementasi yang lain, tulisan menguji prinsip dasar yang menginspirasi ini dalam tempat pertama, struktur dasarnya, dan hubungannya untuk menentukan dengan baik metode-metode dan teknik-teknik OR/MS. Karena itu, makalah ini menunjukkan bahwa pada permalaan Algoritma Dijkstra diinspirasi oleh *Bellman's Principle of Optimality* dan itu, tidak secara mengejutkan, secara teknis ini seharusnya dilhat sebagai sebuah *dynamic programming successive approximation procedure.*

Satu dari implikasi sekarang dari kenyataan ini adalah bahwa dosen-dosen OR/MS dapat memperkenalkan Algoritma Dijkstra sebagai sebuah persamaan algoritma OR/MS yang unggul dan ini berhubungan untuk menentukan dengan metode-metode dan teknik-teknik OR/MS. Pada umumnya, mereka dapat menggabungkan Algoritma Dijkstra dalam silabus-silabus pemrograman dinamik dimana kenyataannya. Ini, kita seharusnya menambahkan, sudah lama menanti-nantikandalam mengajar Algoritma Dijkstra ini adalah paling membangun untuk menarik perhatian mahasiswa untuk fakta bahwa algoritma ini diinspirasi oleh *Bellman's Principle of Optimality*. Lebih jauh, ini penting untuk menunjukkan bahwa ini tidak lebih, tidak kurang daripada sebuah metode untuk menyelesaikan persamaan fungsional pemrograman dinamik untuk masalah jalan terpendek yang diberikan panjan simpul tidak negatif.

Fakta-fakta dasar tentang kedekatan hubungan yang ada adnata Algoritma Dijkstra dan pemrograman dinamik secara pasti didokumentasikan dengan baik pada literatur khusus (contoh. Lawler [1976] dan Denardo [2003]) tetapi tidak dapat diperoleh untuk moyaritas mahasiswa yang memiliki pertemuan pertama dengan pemrograman dinamik dan/atau Algoritma Dijkstra melalui kuliah perkenalan yang memiliki tujuan yang umum dalam OR/MS dan/atau CS.

Secara singkat, tujuan analisis kita pada pemberian Algoritma Dijkstra sebuah perspektif OR/MS oleh pemusatan pada hubungannya dengan metode-metode dan teknik-teknik pemrograman dinamik standar. Ini, kita berharap, akan berkontribusi untuk pemahaman yang penuh dari keduanya bauk dosen dan mahasiswa. Target pembaca utama terdiri atas dua grup – tidak perlu saling terputus: Dosen-dosen mengajar pemrograman dinamik dan/atau Algoritma Dijkstra sebagai topic dalam mata kuliah khusus dan/atau dalam banyak perluasan mata kuliah OR/MS dan CS.

Pada bagian selanjutnya akan dogambarkan struktur dasar dari masalah jalan terpendek yang klasik dan secara singkat menguji masalah ini dari sebuah sudut pandang algoritmik dengan mengarahkan ke dua golongan terkenal dari algoritma- algoritma OR/MS. Satu diinspirasi dari pemrograman linier (*linear programming),* yang lain oleh formulasi-formulasi pemrograman dinamik (*dynamic programming)* dari masalah jalan terpendek tersebut (*shortest path problem).* Lalu mengambarkan Algoritma Dijkstra dan mengidentifikasi golongan dari masalah jalan terpendek ini yang didesign untuk diselesaikan. Terakhir diulas hubungan Algoritma Dijkstra dan pemrograman dinamik.

Ini menarik untuk mencatat bahwa meskipun keterkenalan algoritma tersebut, pada faktannya mungkin dengan tepat karena ini, bertahun-tahun literaturnya tidak hanya telah menyebabkan hasil- hasil yang berguna tetapi juga salah tafsir, salah tafsir dan kesalahan-kesalahan sederhana. Ini dikarenakan pentingnya setuju dengan hasil ini - paling tidak menyebutkannya – di dalam kelas dan di dalam buku bacaan. Secara tepat diulas permasalahan ini dengan *heading Temptations*.

Sehubungan gaya presentasi: berhati-hati diadaptasi sebuah pendekatan semi formal. Ini harus dititikberatkan karena ulasan ini saat membacanya dapat dipertanggungjawabkan untuk sedikit banyak penjelasan formal terperinci dan tafsiran, sama baikmya untuk sebuah perlakuan teknis yang lebihformal dan teliti.
Sejumlah modul interaktif yang berorientasi pendidikan disediakan untuk percobaan dengan Algoritma Dijkstra dan ide yang berhubungan..

2. Masalah-masalah jalan terpendek (Shortest path problems)

Satu dari alas an utama untuk keterkenalan dari Algoritma Dijkstra adalah satu dari algoritma terpenting dan terpakai yang tersedia untuk menghasilkan is that it is one of the most important and useful algorithms available for generating solusi-solusi yang optimal (tepat) untuk golongan yang besar dari masalah-masalah jalan terpendek(*shortest path problems).* Inti dari

golongan dari masalah adalah sangat penting secara teoritis, secara praktis, seperti secara pendidikan.

Tentu saja, ini aman untuk mengatakan bahwa masalah jalan terpendek adalah satu dari masalah-masalah umum yang paling penting dalam bidang-bidang seperti OR/MS, CS dan kecerdasan buatan (AI). Satu dari alasan-alasan untuk ini bahwa pada dasarnya beberapa masalah optimasi kombinatorial dapat diformulasikab sebagai sebuah *shortest path problem*. Jadi, golongan masalah iniluas sekali dan mencakup banyak masalah praktikal yang tidak dapat dilakukan dengan sesungguhnya ("*genuine*") masalah-masalah terpendek.

Golongan baru dari masalah jalan terpendek tulen menjadi sangat penting hari-hari ini dalam hubungan dengan aplikasi- aplikasi praktikal dari system informasi geografi *(Geographic Information Systems (GIS))* seperti komputasi pada tujuan perjalanan. Ini tidak mengejutkan karena bahwa, sebagai contoh, Microsoft memiliki sebuah proyek penelitian pada algoritma-algoritma untuk masalah jalan terpendek.

Apa yang mungkin membuat mahasiswa sedikit terkejut adalah berapa lama sejarah penggunaan matematika untuk masalah yang generik ini menjadi topik dari pekerjaan penelitian yang yang luas yang pada akhirnya menghasilkan metode solusi yang efisien. Namun, pada banyak hal ini tidak sebuah kecelakaan pada semua yang masalah generik ini adalah sangat baik dihubungkan dengan hari-hari dari OR/MS dan komputer elektronik. Kutipan berikut sangat indikatip (Moore [1959, p. 292]):

> Masalah tersebut pertama kali diselesaikan dalam hubungan dengan mesin Claude Shannon penyelesaian jaringan jalan yang ruwet. Ketika mesin ini digunakan dengan jaringan jalan yang ruwet yang memiliki lebih dari satu solusi, seorang pengunjung bertanya mengapa ini tidak dibangun untuk selalu menemukan jalan terpendek. Shannon mencoba menemukan Metode-metode hemat melakukan ini dengan mesin. Dia menemukan beberapa metode yang cocok untuk komputasi analog, dan menghasilkan algoritma-algoritma ini. Beberapa bulan kemudian aplikasi dari ide ini untuk masalah-masalah praktikal dalam komunikasi dan sistem transportasi dianjurkan.

Tujuan dari ulasan ini cukup untuk mempertimbangkan hanya versi "klasik dari masalah jalan terpendek yang umum. Di sana banyak masalah lain.

Pikirkan sebuah masalah yang terdiri dari n > 1 kota {1,2,...,n} dan sebuah matriks yang menggambarkan panjang antara dua kota, jadi D(i,j) menunjukkan panjang yang menghubungkan kota i dan kota j. Ini berarti ada paling banyak satu hubungan antara dua pasang kota. Jaraknya tidak diasumsikan simetris, jadi D(i,j) tidak perlu sama dengan D(j,i). Tujuan adalah menemukan jalan terpendek dari kota h yang diberikan, disebut rumah, untuk kota d, diseebut tujuan. Panjang dari sebuah jalur diasumsikan sama dengan untuk penjumlahan panjang dari hubungan antara kota yang berurutan pada jalur tersebut.

Dengan tidak kehilangan keadaan yang umum, diasumsikan bahwa h=1 dan d=n. Jadi pertanyaan dasar: apa jalan terpendek dari kota 1 ke kota n?

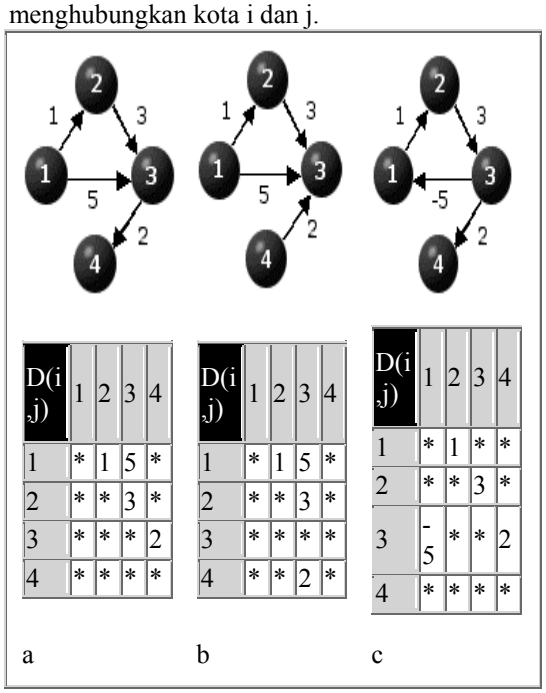
Untuk dapat dengan mudah mengatasi situasi-situasi dimana masalah tersebut tidak dapat dikerjakan dengan mudah (tidak ada jalan dari kota 1 ke kota n), bisa dikembangkan konvensi bahwa jika tidak ada jalur langsung dari kota 1 ke kota j lalu D(i,j) sama dengan berpa saja. Jadi, seharusnya menyimpulkan bahwa panjang jalan terpendek dari fimpul i ke simpul j sama dengan berpa saja implikasinya menjadi bahwa tidak ada jalan dari simpul i ke simpul j.

Tinjau bahwa subjek untuk konvensi ini, sebuah hal dari masalah jalan terpendek dispesifikasi dengan unik oleh panjangnya matriks D. Jadi, matriks ini dapat dipandang sebagai sebuah model yang sempurna untk masalah tersebut.
Sejauh solusi-solusi optimum yang penting, dapat dibedakan menjadi tiga situasi dasar:

- Sebuah solusi optimal yang ada.
- Tidak ada solusi optimal karena tidak ada solusi yang mungkin.
- Tidak solusi yang optimal karena panjang jalur yang mungkin dari kota 1 ke kota n tak terhingga.

Secara ideal, algoritma-algoritma mendesign untuk solusi masalah jalan terpendek seharusnya dapat diatasi dengan tiga kasus ini.

Gambar 1 mengilustrasikan tiga kasus ini. Kota-kota direpresentasikan oleh simpul-simpul dan jarak digambarkan pada simpul dari graf. Dalam tiga kasus In all three cases n=4. Jarak masing-masing matriks juga disediakan. Simbol "*" merepresentasikan nilai berapa saja jadi implikasi dari D(i,j)="*" bahwa tidak ada yang menghubungkan kota i dan j.



D(i,j)	1	2	3	4
1	*	1	5	*
2	*	*	3	*
3	*	*	*	2
4	*	*	*	*

D(i,j)	1	2	3	4
1	*	1	5	*
2	*	*	3	*
3	*	*	*	*
4	*	*	2	*

D(i,j)	1	2	3	4
1	*	1	*	*
2	*	*	3	*
3	-5	*	*	2
4	*	*	*	*

a b c

Gambar 1

Dengan melihat masalah pada gambar 1(a) memeliki jalan optinal yang unik, yaitu x=(1,2,3,4), yang memiliki panjang sama dengan 6. Masalah yang digambarkan dalam gambar 1(b) tidak dapat diselesaikan karena solusinya optimal. Gambar 1(c) mengggambarkan masalah dimana tidak ada solusi yang optimal karrena panjang jalan dari simpul 1 ke simpul 4 dapat dibuat berubah-ubah dengan memutar simpul-simpul 1, 2, dan 3. Setiap putaran akan menurunkan panjang dari jalaur oleh 1.

Tinjau bahwa jika membutuhkan jalur yang dapat diselesaikan dengan mudah, yakni tidak memasukkan putaran, lalu masalah yang digambarkan dalama gambar 1(c) dapat dibatasi. Tentu saja, ini memiliki jalurr optimal yang unik x=(1,2,3,4) yang memiliki panjang sama dengan 6. Pada ulasan ini tidak memaksakan kondisi ini pada bentuk formulasi masalah, yakni berlaku untuk jalur lingkaran sebagai solusi yang dapat dikerjakan yang disediakan bahwa mereka memenuhi pembatas yang lebih diutamakan. Jadi, x'=(1,2,3,1,2,3,4) dan x"=(1,2,3,1,2,3,1,2,3,4) adalah solusi yang dapat dikerjakan untuk masalah yang digambarkan dalam gambar 1(c). Ini adalah cara dalam konteks makalah ini, masalah yang tidak memiliki solusi yang optimal.

Misalkan C={1,2,...,n} merupakan himpunan kota-kota dan setiap kota j dalam C dimisalkan P(j) denote the set of its *immediate predecessors,* and let S(j) denote the set of its *immediate successors,* namely set

$$P(j) = \{k \text{ in } C: D(k,j) < \text{infinity}\} \text{ , } j \text{ in } C \quad (1)$$

$$S(j) = \{k \text{ in } C: D(j,k) < \text{infinity}\} \text{ , } j \text{ in } C \quad (2)$$

Jadi, untuk masalah yang digambarkan dalam gambar 1(a), P(1)={}, P(2)={1}, P(3)={1,2}, P(4)={3}, S(1)={2,3}, S(2)={3}, S(3)={4}, S(4)={}, dimana {} merupakan himpunan kosong.

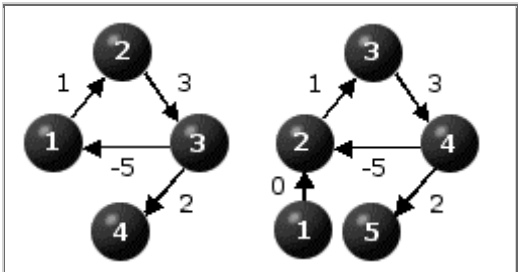
Juga, misalakan NP merupakan himpunan kota-kota yang mempunyai pendahulu-pendahulu yang dekat, dan misalkan NS merupakan himpunann kota-kota yang tidak mempunyai pengganti yang dekat, dimisalakn sebagai berikut :

$$NP = \{j \text{ dalam } C: P(j) = \{\}\} \quad (3)$$

$$NS = \{j \text{ dalam } C: S(j) = \{\}\} \quad (4)$$

Jadi, dalam kasus gambar 1(a), NP={1} and NS={4}. Dengan jelas, jika kota 1 dalam NS dan/atau kota n dalam NP lalu masalah ini tidak dapat diselesaikan..

Untuk alas an-alasan teknis, ini sangat nyaman untuk mengasumsikan bahwa P(1) = {}, yaitu bahwa kota1 tidak mempunyai kota-kota pendahulu yang dekat. Ini adalah sebuah formalitas belaka karena jika kondisi ini tidak dipenuhi, kita dapat menyederhanakan dengan memperkenalakan sebuah *dummy city* dan menghubungkannya dengan kota 1 dengan sebuah sisi dengan panjang 0. Lalu dapat diasumsikan *dummy* kota ini daripada kota 1 sebgai kota *home*. Persolan pemodellan minor ini diilustrasikan dalam gambar 2.



D(i,j)	1	2	3	4
1	*	1	*	*
2	*	*	3	*
3	-5	*	*	2
4	*	*	*	*

D(i,j)	1	2	3	4	5
1	*	0	*	*	*
2	*	*	1	*	*
3	*	*	*	3	*
4	*	-5	*	*	2
5	*	*	*	*	*

a b

Gambar 2

Dua masalah tersebut sama dalam pengertian bahwa masalah dari penemuan sebuah jalan optimal dari kota 1 ke kota 5 dalam gambar 2(a) sama pada masalah dari penemuan sebuah jalan optimal dari kota 1 ke kota 5dalam ganbar 2(b). Ada korespodensi satu satu antara solusi kedua masalah tersebut.
Jadi dengan tidak kehilangan keadaan yang umum aumsikan bahwa P(1)={}.

Antisipasi dari ulasan tersebut pada klasisfikasi masalah jalan terpendek dan persoalan algoritmik yang berhubungan, melihat bahwa makalah ini mengkomputasi panjang jalan terpendek dari simpul 1 ke semua simpul. Ini dapat diatasi masalah *cyclic* dan *acyclic* dan ini tidak dipedulikan apakah jaraknya negatif atau tidak negatif. Fleksibilitas dan kepandaian dalam banyak hal ini tidak hal untuk pemecahan algoritmik tetapihanya sebuah pemikiran dari sebuah fakta bahwa sebgai keadaan diatas tidak meniadakan jalur-jalur cyclic. Dengan kata lain, tidak membatasi jalur yang dapat dikerjakan menjadi mudah*(simple):* membolehkan jalur yang dapat dikerjakan untuk mengandung *cycles*. Ini berarti sekalipun ada hanya terbatas banyak kota, jika jarak dibolehkan negatif dan ada sebuah *cycle* yang memiliki panjang negatif, lalu panjang dari jalan terpendek dapat tak terhingga dari bawah..

Ini mungkin tempat yang cocok untuk mengatakan tentang notasi ganjil dari h=jarak negatif ***(negative distances)*.**

- ✓ Ini harus dititikberatkan bahwa ini tidak sebuah notasi ganjil untuk semua karena istilah "jarak" *(distance)* digunakan sebagai sebuah metafora. Ada banyak masalah praktikal dimana istilah jarak direpresentasikan biaya dan/atau keuntungan, dan dalam konteks ini sangat dasar untuk membicarakan tentang jarak positif dan negatif tanpa kerangka dari masalah yang sama.
- ✓ Ini sehasusnya dirtitikberatkan bahwa istilah "masalah jalan terpendek" *(shortest path problem)* ini sendiri adalah sebuah metafora.Banyak masalah optimasi yang tidak dapat melakukan dengan masalah jalan terpendek yang nyata. Sebgai contoh, masalah rasnsel yang kalsik, masalah pengganti perlengkapan, masalah produksi ukuran, dan masalah perencanaan investasi semuanya memilki representasi jalan terpendek yang sangat sederhana dan dasar (lihat sebagai contoh Evans and Minieka [1992]). Dalam kenyataannya, seperti yang disebutkn diatas, banyak masalah optimasi kombinatorial memiliki solusi yang terbatasyang dapat diformulakan sebagai sebuah masalah jalan terpendek. Pada konteks umum yang luar biasa ini, notasi negatif tersebut tidak ganjil semuanya.
- ✓ Ada beberapa kasus masalah aslinya dicoba diselesaikan untuk diselesaikan memiliki hanya jarak-jarak tak negatif tetapi metode tersebut digunakan untuk menyelesaikan ini secara general menyelesaikan masalah jalan terpendek yang lain (pada udara) yang memiliki jarak yang negatif. Kasus ini sebagai contoh, dalam masalah jaringan dimana fungsi objektif adalah perbandingan dua fungsi (lihat contoh Lawler [1976, pp. 94-97]).
- ✓ Ada banyak situasi praktikal dimana dalam jalur terpanjang lebih menarik daripada jalur terpendek. Sebagai contoh, metode jalur kritis *(Critical Path Method (CPM))* berdasarkan pendeteksian jalur terpanjang pada masalah jaringan (Daellenbach et al [1983], Markland and Sweigart [1987], Hillier and Lieberman [1990], Winston [2004]). Jika kita megaplikasikan trik lama dari perkalian fungsi objektif oleh -1 untuk mentranformasikan sebuah

masalah maksimum menjadi menjadi sebuah masalah minimalisasi yang sama, lalu kita dapat mngalikan semua jarak dengan -1. Maka jika semula jaraknya adalah positif, setelah ditransformasikan semuanya akan menjadi negatif. Masalah minimalisasi yang ekivalen pdada masalah memiliki panjang yang negatif.

Untuk alasan ini, memperbolehkan jarak kota menjadi negatif bukan ide yang aneh. Daripada pemikiran dari fakta bahwa banyak masalah terpendek yang penting memiliki fitur ini. Oleh karena itu, perhatian dalam algoritma-algoritma yang mampu menyelesaikan masalah jalan terpendek yang memiliki jarak diperbolehkan negatif adalah tidak sifat akademikus belaka.

Pada pembahasan berikutnya akan dibahas dua keadaan yang biasa paradigma OR/MS untuk pengembangan algoritma masalah jalan terpendek.

3. Pandangan Algoritmik (*Algorithmic perspective*)

Untuk menghargai peran bahwa Algoritma Dijkstra memainkan dalam analisis dan solusi masalah jalan terpendek dan dimana ini pas menjadi kurikulum OR/MS, ini mengandung pelajaran dan membangun untuk membedakan antara beberapa tipe algoritma yang ada untuk solusi masalah umum yang penting ini. Tentu ada sejumlah cara pengklasifikasian algoritma. Klasifikasi ini OR/MS dan berorientasi metodologi: ini membedakan antara *dynamic programming* dan *linear programming*. Kita titikberatkan bahwa skema klasifikasi yang lain digunakan keduanya dalam literatur ilmu computer dan operasi pencarian (contoh *label-setting* dengan *label correcting methods*, Bertsekas [1991, pp. 67-68], Evans and Minieka [1992, p. 82]). Oleh tanda yang sama, seperti yang kita akan lihat, ada dua kelas yang luas dapat disaring menjadi beberapa sub kelas.

Algoritma pemrograman linier

Ini tidak sulit memformulasikan masalah jalan terpendek sebagai sebuah masalah pemrograman linier. Pada umumnya, pendekatan yang sangat biasa adalah memformulasikan sebagai sebuah *linear network flow problem.* Ini menghasilkan masalah-masalah pemrograman linier yang dikhususkan yang dapat diselesaikan oleh versi efisien dari metode sederhana *(simplex method)* (Ford and Fulkerson [1962], Dantzig [1963], Bertsekas [1991], Evans and Minieka [1992], Ahuja et al [1993], Winston [2004]).

Sebgai contoh, jika masalah jalan terpendek adalah *acyclic* dan memiliki jarak-jarak tidak negatif, ini dapat dipandang dan diformulasikan sebagai *transhipment problem*: Kota 1 adalah sebuah *supply city,* kota n sebgai *demand city* dan kota-kota yang lain {2,...,n-1} adalah *transhipment cities* (kota-kota ini tidak menghasilkan apa-apa dan tidak membuthkan apa-apa). *Total supply-nya* sama dengan *total demand*-nya dan dapat ditetapkan secara berubah-ubah ke satu unit. Jika hasilnya dikatakan sebuah grand piano kecil, pertanyaannya adalah : berapa biaya minimum untuk mengantarkan satu grand piano kecil dari kota 1 ke kota n? Sebuah matiks biaya menspedifikasikan biaya pengantaran satu barang (grand piano kecil) dari kota i dengan langsung ke kota j.

Secara lebih umum, masalah jalan terpendek tersebut dapat diformulasikan sebagai sebuah *minimum cost network flow problem.* Model pemrograman linier adalah sebagai berikut. Dimisalkan

x_{ij} = Kuantitas (aliran) dikirim sepanjang lintasan dari kota i ke kota j; i,j=1,2,...,n (5)

Lalu masalah aliran jaringan masalah minimum adalah sebagai berikut:

$$\text{Min } S_{ij}c_{ij}x_{ij} \quad (6)$$

s.t.

$$S_k x_{jk} - S_i x_{ij} = s_j \text{ , } j=1,2,...,n \quad (7)$$

$$x_{ij} >= 0 \text{ ; } i,j=1,...,n \quad (8)$$

Dimana s_j merupakan persediaan pada simpul (kota) j pada jaringan dan koefisien biayat c_{ij} direpresentasikan biaya pengiriman satu barang dari aliran sepanjang sisi yang menghubungkan simpul i ke simpul jrepresents. S diasumsikan untuk menjumlahkan indeksnya atas jarak {1,...,n}, sebgai contoh $S_k x_{jk} = x_{j1} + ... + x_{jn}$. Catatan bahwa (7) sebuah pembicaraan dari batasan aliran: pada setiap simpul total aliran masuk= total aliran keluar.

Pada kasus dari masalah jalan terpendek tersebut ditetapkan $c_{ij} = D(i,j)$, s(1)=1, s(n)=-1 dan s(j)=0 untuk j=2,...,n-1.

Tiada gunanya untuk mengatakan, karena *LP software* sangat dapat menembus dalam OR/MS, paradigma LP untuk masalah jalan terpendek tersebut memiliki sebuah daya tarik yang utama untuk akademikus, murid-murid dan pelaksana-pelaksana.

Algoritma pemrograman dinamik

Algoritma-algoritma dari tipe ini diinspirasi oleh Bellman's [1957, p. 83] *Principle of Optimality* yang terkenal:

> Sebuah kebijakan yang optimal memiliki sifat yang *initial state* dan *initial decision* apa saja, sisa keputusan tersebut harus terdapat sebuah kebijakan optimal dengan hal untuk keadaan hasildari keputusan pertama.

Dipedomani oleh prinsip ini, hal pertama yang dilakukan adalah mengumumkan masalah jalan terpendek (menemukan sebuah jalur optimal dari kta 1 ke kota n) dengan melekatkan ini dalamsebuah hubungan masalah (memnemukan jalur optimal dari kota i ke kota j, untuk j=1,2,3,...,n). Jadi dalam sebuah masalah tipe pemrograman dinamik,dimisalkan

f(j) := panjang dari jalur terpendekdari simpul 1 ke simpul j, j=1,2,3,...,n. (9)

Ini penting menitikberatkan tujuan ulasan ini menentukan menentukan nilai dari f(n). Nilai dari {f(j), j=1,2,...,n-1} tersebut tidak perlu dimasukkan karena kita berminat kepada nilainya, tetapi pertama dan terutama karena ini cara pemrograman dinamik bekerja: layak untuk menentukan nilai dari f(n), ini mungkin untuk menenukan nilai dari f(j) untuk j=1,2,...,n-1 dengan baikl.

Pada beberapa kasus, menggunakan definisi di atas dari f(j), berikut adalah satu dari implikasi terdekat dari prinsip ini dalam konteks masalah jalan terpendek:

Corollary 1

f(j) = D(k,j) + f(k) , untuk beberapa kota k dalam P(j) (10)

untuk beberapa kota j seperti P(j) != {}, dimana != merupakan berarti tidak sama.

Tentu, bicara secara algoritmik, hasil ini tidak memuaskan sama sekali karena ini tidak diidentifikasi secara tepat nilai dari k supaya (10) dipenuhi. Ini menjamin bahwa ada sebuah k dan sebuah elemen dari himpunan P(j).

Namun, secara konseptual hanya sebuah hambatan yang kecil. Setelah semua, f(j) merupakan jarak terpendek *(shortest distance)* dari simpul 1 ke simpul j dan karena itu jelas bahwa k yang aneh tersebut pada sebelah tangan kanan dari (10) dapat diidentifikasi dengan membuat sisi tangan kanan dari (10) sekecil-kecilnya. Ini menyebabkan:

Corollary 2

f(j) = min {D(k,j) + f(k): k dalam P(j)} , jika P(j) != {}. (11)

f(j) = jumlah tak terhingga, jika P(j) = {} dan j > 1. (12)

f(1) = 0 , (diasumsikan bahwa P(1)={}). (13)

Ini adalah persamaan fungsional pemograman dinamik untuk masalah jalan terpendek.

Ini seharusnya dititikberatkan, disini dan dimana pun, persamaan fungsional pemrograman dinamik tidak merupkan sebuah algoritma. Ini menetapkan sifat yang pasti bahwa fungsi f didefinisikan dalam (9) harus terpenuhi. Tentu saja, dalam konteks dari *Corollary 2* tersebut, ini merupakan sebuah kondisi optimal yang penting. Poin ini kadang-kadang dihargai oleh murid pada pertemuan pertama mereka dengan pemrograman dinamik. Rupanya ini sebuah pemikiran dari fakta bahwa dalam banyak contoh-contoh buku bacaan, pengertian dari algoritma digunakan untuk memecahkan persamaan fungsional adalah kebanyakan salinan dari persamaan ini sendiri.

Kesimpulan, pemrograman dinamik untuk masalah jalan terpendek menghasilkan disain untuk menetapkan nilai nilai dari {f(j)} didefenisikan dalam (9) oleh penyelesaian persamaan fungsional pemrograman dinamik (11)-(13).

Ini mungkin untuk memformulasikan sebuah model pemrograman linier untuk masalah jalan terpendek yang dengan baik sekali lebih berhubungan ke persamaan fungsional pemrograman dinamik (11)-(13). Petunjuk: membangun rangkap dari (6)-(8) (lihat Lawler [1976, pp. 78-82] untuk lebih detail mengenai hubungan antara formula ini dan sejumlah

algoritma-algoritma untuk masalah jalan terpendek tersebut).

Kesimpulan, banyak pendekatan algoritmik yang berbeda untuk solusi masalah jalan terpendek (contoh pemrograman linier, pemrograman dinamik). Pertanyaan pertama yang dating pada saat ini dengan nyata: Apakah Algoritma Dijkstra sebuah algoritma yang diinspirasi pemrograman dinamik? Sebuah algoritma diinspirasi aliran jaringan? Atau mungkin sebuah algoritma dari tipe yang lain pada keseluruhannya?

Berikut ini diambil dari sebuah iklan IBM dari paket OSL mereka mungkin menyediakan jawaban :

> Semut tidak memerlukan Algoritma Dijkstra' untuk menemukan jalan terpendek dalam bepergian. Mereka selalu menemukan jalan mereka, meyeleksi rute tercepat, menghindari hambatan-hambatan, memaksimumkan hasil dengan usaha yang minimum. Mendekati pendekatan yang sempurna dari sumberdaya dengan keuntungan maksimum; mencapai pekerjaan raksasa dengan kerjasama dan *teamwork* yang tiada bandingannya. Apakah tidak mungkin jika *software* dapat seperti itu?
> OR/MS Today, 29(1), p. 37, 2002

Hal algoritmik yang lain yang harus dialamatkan adalah tipe dari masalah bahwa sebuah algoritma yang diberikan dapat menyelesaikan. Antisipasi dari inis, beberapa kelas masalah jalan terpendek:

- ✓ Masalah *Acyclic*: tidak ada sub jalur yang dapat diselesaikan yang dimulai dan diakhiri pada kota yang sama.
- ✓ Masalah *Cyclic*: ada sub jalan yang dapat diselesaikan yang dimulai dan diakhiri pada kota yang sama.
- ✓ *Masalah jarak tidak negatif*: jarak antar kota tidak negatif. Itu , D(i,j) >=0 untuk semua i dan j.
- ✓ *Masalah jalan negatif*: beberapa jarak antar kota negatif. Yaitu, D(i,j) < 0 untuk beberapa i dan j.
- ✓ Masalah *cyclic* tidak negatif: masalah *cyclic* dengan sifat yang panjang dari setiap *cycle* tidak negatif(contoh gambar 3).
- ✓ Masalah *cyclic* negatif: masalah *cyclic* dengan sifat paling sedikit terdapat sati *cycle* yang negatif(contoh gambar 2(b)).

Catatan bahwa sifat Cyclic/Acyclic ditentukan oleh himpunan {P(j)} sedangkan sifat jarak negatif atau tidak negatif ditentukan oleh nilai dari jarak matriks D.

4. Algoritma Dijkstra

Dari sebuah hal teknis belaka dari pandangan Algoritma Dijkstra dapat digambarkansebgai sebuah prosedur iteratif yang diinspirasi oleh (10) yang berkali-kali mencoba untuk mengemabangkan sebuah perkiraan awal {F(j)} dari nilai yang tepat dari {f(j)}. Perkiraan awal sederhana F(1)=0 dan F(j)=tak terhingga untuk j=2,...,n. Rinciannya sebagai berikut.

Setiap kota diproses dengan tepat sekali berdasarkan sebuah perintah untuk dispesifikasi segera. Kota 1 diproses pertama. Sebuah catatan (himpunan) disimpan pada kota-kota tersebut yang belum diproses, ini disebut dengan U. Jadi pada awalnya U = C = {1,...,n}. Ketika kota k diproses tugas berikutnya dilakukan:

Memperbaharui F: himpunan F(j) = min{F(j),D(k,j) + F(k)}, untuk semua j dalam U /\ S(k) (14)

dimana A/\B merupakan pertemuan dari himpunan A dan B. Menimbulkan S(j) sebagai himpunan dari pengganti terdekat dari kota j. Jadi, ketika kota k diproses, nilai {F(j)}dari pengganti terdekatnya yang belum diproses diperbaruhui sesuai dengan (14).

Untuk melengkapi deskripsi informal dari algoritma tersebut, ini hanya perlu untuk menspesifikasikan perintah tersebut dimana kota tersebut diproses. Ini tidak sulit: kota berikutnya yang diproses adalah satu yang memiliki nilai F(j) terkecil dari semua kota yang belum diproses:

Memperbaharui k: k = arg min {F(j): j dalam U} (15)

Menimbulkan U sebagai himpunan kota-kota yang belum diproses. Jadi, pertama kalinya U = {1,...,n} dan setelah kota k diproses, ini dengan segera dihapus dari U. Secara resmi

Memperbaharui U: U = U\{k} (16)

Kita melihat dalam penyampaian bahwa alasan bahwa Algoritma Dijkstra dipandang sebgai sebuah metode yang tamak dalam kaidah ini dikembangkan, (15), untuk menyeleksi kota berikutnya untuk diproses: kota berikutnya untuk diproses adalah salah satu yang terdekat ke kota 1 yang belum diproses.

Jadi pengambilan tiga bagian bersama-sama, ini sebuah deskripsi lebih formal dari Algoritma Dijkstra:

Deskripsi dari Algoritma Dijkstra

Algoritma ini bekerja dengan menyimpan setiap simpul *v* biaya *d*[*v*] dari jalur terpendek yang ditemukan sangat jauh antara *s* dan *v*. Pada awalnya, nilai ini adalah 0dari simpul sumber *s* (*d*[*s*]=0), dan jumlah yang tak berakhir dari semua simpul yang lain, merepresentasikan fakta bahwa kita tidak tahu beberapa jalur utama untuk simpul-simpul itu (*d*[*v*]=∞ dari setiap *v* dalam *V*, kecuali *s*). Ketika algoritma tersebut berakhir, *d*[*v*] akan menjadi biaya dari jalur terpendek dari *s* ke *v* — atau tak terhingga, jika tidak ada jalur yang ada.

Operasi dasar dari Algoritma Dijkstra adalah pengendoran sisi ***(edge relaxation)***: jika ada sebuah sisi dari *u* ke *v*, lalu jalur terpendek yang diketahui dari *s* ke *u* (*d*[*u*]) dapat diperluas ke sebuah jalur dari *s* ke *v* dengan penambahan sisi (*u*,*v*). Jalur ini akan memiliki panjang *d*[*u*]+*w*(*u*,*v*). Jika ini kurang dari *d*[*v*] yang baru, kita dapat menggantikan nilai terbaru dari *d*[*v*]dengan nilai baru tersebut. Pengendoran sisi diaplikasikan sampai semua nilai *d*[*v*] merepresentasikan biaya dari jalur terpendek dari *s* ke *v*. algoritma tersebt diorganisasikan jadi setiap simpul (*u*,*v*) dikendurkan hanya sekali, ketika *d*[*u*] mencapai nilai akhirnya.

Notasi pengendoran ***(relaxation)*** datang dari sebuah analogi antara kalkulasi dari jalan terpendek dan panjang dari sebuah *helical tension spring*, yang tidak didisain untuk kompresi. Pada awalnya , biaya dari jalur terpendek sebuah taksiran yang terlalu tinggi, disamakan dengan pegas yang berkepanjangan. Sebagai jalur terpendek yang ditemukan, penaksiran biaya diturunkan, dan pegas dikendorkan. Akhirnya, jalur terpendek, jika satu tersedia, diemukan dan pegas telah dikendorkan ke panjangnya yang kendor.

Algoritma tersebut mempertahankan dua himpunan dari simpul *S* dan*Q*. Himpunan *S* mengandung semua simpul untuk yang kita mengetahui nilai *d*[*v*] adalah sudah menjadi biaya dari jalur terpendek dan himpunan *Q* mengandung semua simpul yang lain. Himpunan *S* dimulai dari kosong, dan dalam setiap langkah setiap simpul dipindahkan dari *Q* ke *S*. Simpul ini diplilih sebagai simpul dengan nilai terkecil dari *d*[*u*]. Ketika *u* dipindahkan ke *S*, algoritma tersebut mengurangi setiap sisi yang keluar (*u*,*v*).

Pseudocode

Pada algoritma berikut, u := Extract_Min(Q) mencari dari simpul *u* dalam himpunan simpul *Q* yang memiliki paling sedikit nilai *d*[*u*]. Simpul tersebut dihilangkan dari himpunant *Q* dan dikembalikan ke *user*.

```
1  function Dijkstra(G, t, s)
2      for each vertex v in V[G]
//Inisialisasi
3            d[v] := infinity
4            previous[v] :=
undefined
5      d[s] := 0
//Jarak dari s ke s
6      S := empty set
7      Q := V[G]
// Himpunan dari semua simpul
8      while Q is not an empty
set              // Algoritmanya
9             u := Extract_Min(Q)
10            S := S union {u}
11            for each edge (u,v)
outgoing from u
12                  if d[u] +
w(u,v) < d[v]      //
Pengendoran (u,v)
13                        d[v]
:= d[u] + w(u,v)
14
previous[v] := u
```

Jika kita hanya tertarik pada jalur terpendek antara simpul *s* dan *t*, kita dapat mengakhiri pencaria pada baris ke 9 jika *u* = *t*. sekarang kita dapat membaca jalan terpendek dari *s* ke *t* dengan pengulangan:

```
1 S := empty sequence
2 u := t
3 while defined previous[u]
4       insert u to the
beginning of S
5       u := previous[u]
```

Sekarang urutan *S* adalah list dari simpul-simpul yang merupakan salah satu dari ja;ur terpendek dari *s* ke *t*, atau urutan yang kosong jika tidak ada jalur.

Masalah yang lebih umum akan ditemukan untuk semua jalur terpendek antara *s* dan *t* (mungkin ada beberapa yang berbeda dengan panjang yang

sama). Sebagai pengganti penyimpan hanya sebuah simpul *single* pada setiap masukan dari sebelumnya [] kita akan menyimpan semua simpul yang memenuhi kondisi pengurangan. Sebgai contoh, jika *r* dan *s* menghubungkan *t* dan keduanya melalui jalan terpendek yang berbeda menuju *t* (karena biaya simpul sama dalam kedua kasus tersebut), lalu kita dapat menambahkan *r* dan *s* ke sebelumnya[t]. Ketika algoritma tersebut lengkap, struktur data sebelumnya[] sebenarnya akan menggambarkan sebuah graf yang sub himpunannya dari graf awal dengan beberapa sisi yang dihilangkan. Kunci sifat ini akan menjadi yang jika algoritma tersebut dijalankan dengan beberapa simpul awal, lalu setiap jalur dari simpul yang satu ke simpul yang lain dalam graf yang baru tersebut menjadi jalur terpendek antara simpul-simpul itu dalam graf awal, dan semua jalur yang panjangnya dari graf awal akan disajikan dalam graf yang baru. Lalu sebenarnya untuk menemukan semua jalur pendek antara dua simpul yang diberikan , kita akan menggunakan jalur yang ditemukan algoritma pada graf yang baruwe,seperti *depth-first search*.

Waktu penjalanan

Kita dapat memperlihatkan waktu pengeksekusian Algoritma Dijkstra pada sebuah graf dengan sisi-sisi *E* dan simpul-simpul *V* sebagai sebuah fubgsi dari |*E*| dan |*V*| menggunkan notasi Big-O.

Implementasi paling singkat dari Algoritma Dijkstra menyimpan simpul-simpul dari himpunan Q dalam sebuah list berkait biasa atau *array*, dan operasi Extract-Min(*Q*) disederhanakan sebuah pencarian linier melalui semua sisi-sisi dalam Q. Pada kasus ini, waktu pengeksekusian adalah $O(V^2)$.

Untuk *sparse graphs*, bahwa, graf dengan lebih sedikit sisi V^2, Algoritma Dijkstra dapat diemplementasikan lebih efisien dengan menyimpan graf tersebut dalam bentuk list yang berdekatan dan menggunakan sbuah *binary heap* atau *Fibonacci heap* sebagai sebuah antrian prioritas *(priority queue)* untuk mengimplementasikan fungsi Extract-Min. Dengan sebuah *binary heap*, algoritma tersebut memerlukan waktu $O((E+V)\log V)$, dan *Fibonacci heap* memperbaiki ini ke $O(E + V\log V)$.

5. Kesimpulan

Algoritma Dijkstra melingkupi dengan luas literatur OR/MS dan CS ;dan adalah sebuah topik popular dalam pengenalan buku bacaan OR/MS. Untuk alasan sejarah , algoritma tersebut dimasukkan sebagai metode ilmu komputer. Tetapi telah ditunjukkan dalam ulasan ini, algoritma ini memiliki sumber OR/MS yang sangat kuat.

Tentu saja, salah satu dari tujuan ulasan ini adalah untuk menunjukkanbahwa situasi ini dapat diubah dengan OR/MS dan juga dengan CS.

Pengembang-pengembang *OR/MS courseware* dianjurkan untuk memikirkan kembali bagaimana cara mereka memperlakukan algoritma ini dan bagaimana mereka menghubungkan ini dengan metode dan teknik OR/MS yang lain. Pada umumnya, algoritma ini dapat digunakan untuk mengilustrasikan penyebaran metode-metode aproksimasi yang berturut-turut oleh pemrograman dinamik.

Daftar Pustaka

[1] Bellman, R. (1957), *Dynamic Programming,* Princeton University Press, Princeton, NJ.

[2]Bertsekas, D. (1991), *Linear Network Optimization,* The MIT Press, Cambridge, MA.
Ford, L.R. and Fulkerson, D.R. (1962), *Flows in Networks,* Princeton University Press, Princeton, NJ.

[3]Brassard, G. and Bratley, P. (1988) *Algorithmics,*Prentice-Hall, Englewood Cliffs, NJ.

[4]Dantzig, G.B. (1960), On the shortest path route through a network, *Management Science,* 6, 187-190.

[5]Dantzig, G.B., (1963), *Linear Programming and Extensions,* Princeton University Press, Princeton, NJ.

[6]Dantzig, G.B. and N.M Thapa, (2003), *Linear Programming 2: Theory and Extensions,* Springer Verlag, Berlin

[7]Denardo, E.V. and Fox, B.L. (1979), Shortest-route methods: 1. Reaching, pruning, and buckets, *Operations Research,* 27, 161-186

[8]E. W. Dijkstra: *A note on two problems in connexion with graphs*. In: *Numerische Mathematik*. 1 (1959), S. 269–271

[9]Evans, J.R. and Minieka, E. (1992), *Optimization Algorithms for Networks and Graphs,* Marcel Dekker, NY.

[10]Gass, S.I. and Harris, C.M. (1996), *Encyclopedia of Operations Research and management Science,* Kluwer, Boston, Mass.

[11]Lawler, E.L. (1976), *Combinatorial Optimization: Networks and Matroids,* Holt, Rinehart and Whinston, NY.

[12]Markland, R.E. and J.R. Sweigart, (1987), *Quantitative Methods: Applications to Managerial Decision Making,* John Wiley, NY.
Winston, W.L. (2004), *Operations Research Applications and Algorithms,* Fourth Edition, Brooks/Cole, Belmont, CA.

[13]Microsoft Shortest Path Algorithms Project: research.microsoft.com/research/sv/SPA/ex.html

[14]Moore, E.F. (1959), The shortest path through a maze, pp. 285-292 in *Proceedings of an International Synposium on the Theory of Switching (Cambridge, Massachusetts, 2-5 April, 1957),* Harvard University Press, Cambridge.

[15]Hillier, F.S. and Lieberman, G.J. (1990) *Introduction to Operations Research,* 5th Edition, Holden -Day, Oakland, CA.

[16]Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 24.3: Dijkstra's algorithm, pp.595–601.

[17]*http://en.wikipedia.org/wiki/Dijkstra's_algorithm.htm*. Tanggal akses: 28 Desember 2006 pukul 09:00.