Berbagai Variasi Teknik Program Dinamis dalam Penyelesaian Masalah

Afrizal Fikri / 13513004[1]
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia
[1]afrizal_f@students.itb.ac.id

***Abstraksi*—Masalah di dunia tidak ada habisnya. Banyak algoritma pun dikembangkan untuk menyelesaikan masalah tersebut. Tapi beragam masalah yang penting di dunia ternyata tergolong dalam NP problem. Tidak ada algoritma yang ditemukan mangkus dalam penyelesaian permasalahan ini. Karenanya bermacam teknik optimasi dikembangkan untuk mereduksi pemakaian waktu dari solusi-solusi. Diantaranya adalah program dinamis.**

***Kata Kunci*—NP-Complete, Program Dinamis, polinomial**

I. PENDAHULUAN

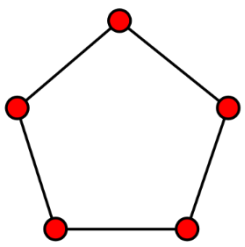
Dari sekian banyak teknik algoritma yang digunakan, program dinamis adalah salah satu solusi yang memberi hasil optimum bahkan pada kasus terburuk sekalipun. Hal ini karena program dinamis tidak bergantung pada fungsi heuristik yang spekulatif.

Meski demikian, alokasi memori yang cukup besar menjadi tantangan tersendiri untuk menggunakan teknik program dinamis ini. Sehingga untuk alasan praktis, penggunaan program dinamis dapat dipertimbangkan lebih jauh. Dan juga secara rata-rata, solusi pendekatan (*approximate solution*) juga lebih cepat.

A. Graf dan Sirkuit

Graf yaitu sebuah himpunan obyek-obyek yang saling terhubung[4]. Graf tersusun atas simpul (obyek) dan sisi (hubungan antar obyek). Graf terdiri atas graf berarah maupun graf tidak berarah. Pada graf berarah, hubungan obyek hanya terjadi searah saja. Sedangkan pada graf tidak berarah dapat dianggap sebagai graf dengan dua arah.

Sirkuit yaitu perpindahan melalui rangkaian simpul yang berakhir pada simpul awal. Sirkuit dapat terjadi pada graf berarah maupun tidak berarah.



Gambar 1.1 Contoh sirkuit pada graf tidak berarah

B. Permasalahan keputusan

Permasalahan keputusan adalah permasalahan yang jawabannya ya dan tidak[1]. Permasalahan ini disebut juga masalah pengecekan atau verifikasi. Kurang lebih untuk satu permasalahan tertentu kita diberi sebuah usulan solusi, kita diminta menentukan apakah solusi yang ditawarkan menjawab permasalahan berikut atau tidak.

Bentuk lain dari masalah ini yaitu masalah predikat atau keanggotaan dari satu himpunan. Yaitu diberikan sebuah predikat keanggotaan himpunan tertentu, tentukan apakah sebuah entitas termasuk anggota himpunan berikut atau bukan.

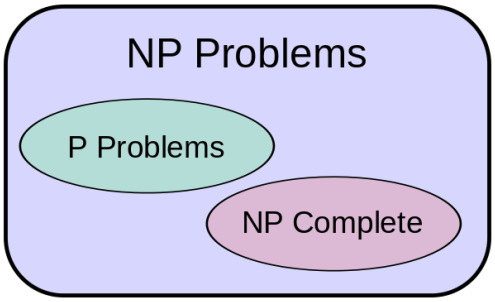
Kebanyakan permasalahan yang biasa ditemui pada dasarnya dapat direduksi menjadi permasalahan keputusan yang ekuivalen. Sebagian yang lain tidak dapat ditentukan jawaban masalah keputusannya, atau biasa dikenal dengan *undecidable problem*[2].

C. Permasalahan P, NP, dan NP-Complete

Permasalahan P adalah jenis permasalahan yang mampu diselesaikan secara mangkus dalam orde waktu polinomial. Sedangkan permasalahan NP adalah permasalahan yang mampu solusinya mampu diverifikasi dalam orde waktu polinomial[1]. Pengertian di atas berkaitan dengan sifat dari masalah keputusan.

Diberikan sebuah permasalahan, lalu diberikan kemungkinan solusinya entah dari mana asalnya. Pada permasalahan NP, akan ada sebuah fungsi verifikasi *V* yang mampu melakukan pengecekan solusi dalam orde polinomial.

Semua permasalahan yang mampu diselesaikan dalam orde polinomial pasti mampu diverifikasi dalam orde polinom juga. Minimal fungsi verifikasi akan langsung membenarkan jawaban solusi. Hal ini karena pencarian solusi sudah pasti akan memberikan jawaban benar. Maka, dapat disimpulkan bahwa masalah P juga dapat diverifikasi solusinya dalam orde polinomial. Atau dengan kata lain P adalah himpunan bagian dari NP ($P \subseteq NP$).



Gambar 1.2 Ilustrasi hubungan P, NP, dan NP-Complete
Sumber :
http://upload.wikimedia.org/wikipedia/commons/thumb/b/bc/Complexity_classes.svg/640px-Complexity_classes.svg.png

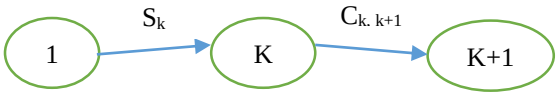
Diantara masalah NP ada jenis NP yang khusus yaitu NP-Complete. Permasalahan NP-Complete adalah permasalahan NP yang masalah NP lainnya dapat direduksi menjadi masalah tersebut dalam orde waktu polinomial. NP-Complete disebut juga dengan NP-Hard meskipun tidak berarti bahwa NP-Hard termasuk masalah yang sulit diselesaikan.

D. Program Dinamis

Program dinamis yaitu metode penyelesaian masalah dengan memecah masalah menjadi tahap-tahap yang lebih kecil yang berhubungan dengan masalah awal[3]. Program dinamis berguna ketika menghadapi masalah yang memiliki lebih dari satu kemungkinan keputusan dalam tiap tahap pencarian.

Karakteristik masalah yang mampu diselesaikan dengan program dinamis yaitu persoalan mampu dipecah menjadi upa-masalah yang serupa (*optimal substructure*) dan terdapat banyak upa-masalah yang digunakan berulang-ulang pada banyak upa-masalah lain yang lebih besar (*overlapping subproblem)*.

Dalam program dinamis digunakan prinsip optimalitas. Yaitu jika sampai tahap *k* solusi optimal, maka solusi optimal pada tahap ke *k+1* adalah solusi sampai tahap *k* digabung (bisa ditambah, dikurangi, dibandingkan atau lainnya tergantung permasalahan) dengan solusi optimal dari tahap *k* ke tahap *k+1*.



Gambar 1.3 Ilustrasi prinsip optimalitas

Permodelan upa-masalah dalam program dinamis menggunakan konsep status masalah. Status masalah yaitu parameter dari masalah yang akan dipecahkan saat ini. Dengan program dinamis, masalah dengan status tertentu akan membutuhkan solusi dari (beberapa) masalah serupa dengan status yang berbeda sebagai dasar pengambilan keputusan untuk menghasilkan hasil pada status sekarang yang optimal.

Maka, program dinamis juga dapat dituliskan dengan bentuk fungsi rekursif. Fungsi ini mengambil hasil optimal dengan memanggil dirinya sendiri (rekursif) dengan parameter dari upa-masalahnya.

E. Travelling Salesman Problem

Masalah Travelling Salesman Problem (TSP) adalah masalah penentuan sirkuit terpendek yang melewati semua simpul dan kembali ke simpul awal. Masalah ini merupakan masalah yang serupa dengan *Hamiltonian Circuit Problem* (HCP).

TSP (dan juga HCP) merupakan masalah yang tergolong NP-Complete. Sehingga penyelesaian yang mangkus pada masalah ini akan menyebabkan semua NP problem dapat diselesaikan secara mangkus, atau dengan kata lain P = NP. Hal ini karena TSP memiliki sifat yang umum dimiliki oleh masalah NP lain. Yaitu urutan pemilihan simpul dalam graf.

F. Boolean Satisfiability Problem

Masalah Boolean Satisfiability Problem (SAT) adalah masalah pengecekan apakah pada satu klausa logika ada nilai yang memenuhi agar hasil klausa tersebut bernilai benar.

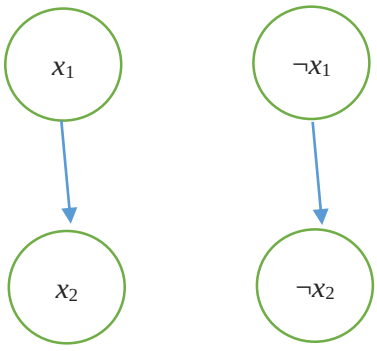
Misal untuk variabel x_1, x_2, dan x_3, tentukan apakah klausa

$$(x_1 \vee \neg x_2) \wedge (\neg x_1 \vee x_2 \vee x_3) \wedge \neg x_1$$

satisfiable atau tidak. Verifikasi dapat dilakukan dengan cepat dalam $O(n)$. Meski begitu

SAT merupakan masalah yang tergolong NP-Complete. Sifat yang umum dari SAT yaitu nilai kebenaran atau fungsi keanggotaan dari masing-masing variabel. Sampai saat ini belum ada algoritma yang mangkus ditemukan pada masalah ini, kecuali pada SAT bervariabel 1 dan 2 (1SAT dan 2SAT).

Pada 2SAT, penyelesaian polinomial dapat dilakukan dengan pengecekan rute yang menyebabkan kontradiksi. Yaitu untuk suatu klausa, buat graf yang merepresentasikan hubungan implikasi antara variabel yang berhubungan. Misal, untuk $x_1 \vee \neg x_2$ ekuivalen dengan $x_1 \rightarrow x_2$ dan $\neg x_2 \rightarrow \neg x_1$. Maka buat graf berarah dari x_1 ke x_2 dan $\neg x_2$ ke $\neg x_1$. Lalu cek apakah ada rute yang kontradiksi (x_1 ke $\neg x_1$ atau sebaliknya atau x_2 ke $\neg x_2$ atau sebaliknya).



Gambar 1.4 graf klausa dari $x_1 \vee \neg x_2$

II. PENDEFINISIAN STATUS

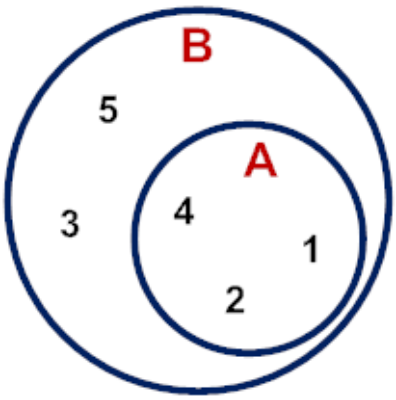
Telah dibahas di atas bahwa program dinamis menggunakan status untuk mendefinisikan langkah penyelesaian masalah. Status yang digunakan bisa bermacam-macam. Pada kasus bilangan Fibonacci, karena masalah sangat sederhana, status hanya memuat urutan bilangan Fibonacci. Tapi pada beberapa kasus, status yang redefinisi dapat lebih dari lima dan cara penghitungan menjadi jauh lebih kompleks.

Pada masalah NP, kebanyakan solusi yang ditawarkan adalah dengan pencarian menyeluruh (*complete search*) dan sedikit optimasi dengan pemangkasan heuristik, seperti *branch and bound* dan A*. Ada juga solusi dengan membatasi lingkup persoalan sehingga solusi yang dihasilkan tidak pasti, tapi hanya perkiraan (*approximate solution*). Solusi dengan cara yang demikian cukup efektif secara statistik tapi untuk kasus terburuk solusi–solusi tersebut tidak jauh berbeda dengan solusi *bruteforce*.

Beberapa pendekatan yang biasa digunakan pada status masalah PD yaitu *set coverage, constraint bounded, dan maximum interval*.

A. Set Coverage

Andaikan pada masalah tertentu terdapat sebuah Graf G(E,V). Dari graf G akan diambil sebuah upagraf G' sehingga solusi optimal G memuat solusi optimal dari upagraf G'. Maka masalah tersebut memenuhi kriteria masalah program dinamis yaitu *optimal substructure.*



Gambar 2.1 Ilustrasi upa-himpunan (subset)

Status dari masalah yang diselesaikan memuat himpunan simpul dari graf sekarang. Model status demikian disebut juga dengan teknik *set coverage*. Himpunan simpul bisa dimodelkan dengan apapun, baik dengan list, heap, atau lainnya.

Tapi untuk memudahkan parametrisasi, biasanya set dimodelkan dengan himpunan bit yang menyatakan simpul tertentu ada atau tidak. Teknik demikian dikenal juga dengan *bitmasking.*

Misal dari G(E,V), terdefinisi himpunan simpul V = {1, 2, 3, 4}. Upagraf sekarang G'(E',V') berisi simpul V' = {1, 3, 4}. Maka bitmasking sekarang yaitu merepresentasikan kehadiran hanya simpul 1, 3 dan 4 saja, yaitu bit pertama, ketiga, dan keempat dari kanan. Sehingga *bitmask* yang merepresentasikan *vertices set* yaitu **1101**.

Permodelan *set coverage* bisa digunakan untuk menyelesaikan masalah TSP. Karena tiap rute bisa dibentuk dari bermacam subrute dari subgrafnya. Maka langkah optimal bisa dicari dari subrutenya.

Perhatikan jika kita ingin mencari sirkuit satu graf, maka kita perlu mencari lintasan optimal untuk tiap graf untuk ditambah lintasan dari simpul awal ke simpul akhir. Lalu satu lintasan juga tergantung pada sublintasan ditambah sisi dari simpul akhir ke simpul baru.

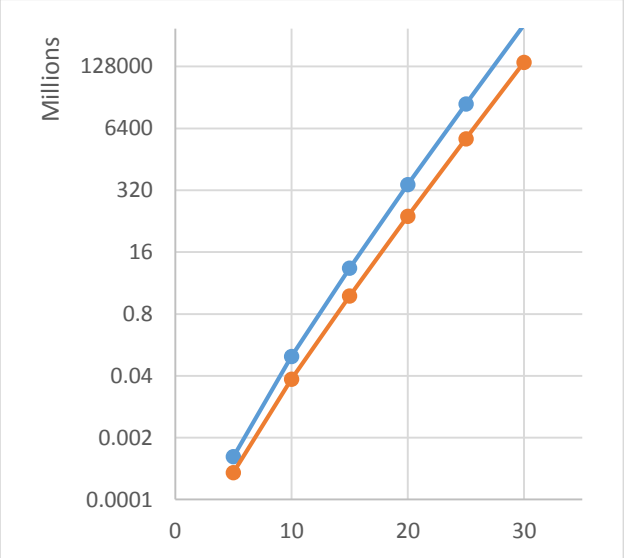
Pencarian lintasan optimal dalam relasi rekurens :

$$f(i,S) = \min_{j \in S}\{f(j,S-j) + e_{i,j}\} \qquad (1)$$

Sedangkan sirkuit optimal dapat dicari dengan :

$$g(i) = \min_{k \in S}\{f(k,S-k) + e_{k,i}\} \qquad (2)$$

dengan i adalah simpul awal, $e_{i,j}$ adalah panjang sisi dari i ke j dan S adalah himpunan simpul dari upagraf sekarang. Solusi akhir dapat dicari dengan mencari minimal untuk semua $g(i)$.



Gambar 2.2 Perbandingan performa algoritma Held-Karp (jingga) dengan algoritma *bruteforce* (biru)

Algoritma ini dikembangkan oleh Held-Karp sehingga disebut juga algoritma Held-Karp. Algoritma ini memiliki kompleksitas total waktu $O(n^2 2^n)$. Bisa dioptimasi lagi dengan menggunakan struktur data adjacent list untuk representasi sisi dan dicari menggunakan pencarian biner (*binary search*). Sehingga kompleksitas total waktunya mencapai $O(n\,2^n \log n)$. Meskipun kompleksitas total masih mencapai eksponensial, tapi performanya lebih

baik dibandingkan *bruteforce* yang mencapai $O(n!)$.

Masalah lain yang bisa diselesaikan dengan *set coverage* ini adalah SAT problem. Fungsi dari himpunan simpul adalah nilai kebenaran variabel. Jadi nilai *bitmask* yang di sini berubah fungsi dari fungsi keanggotaan menjadi nilai kebenaran variabel.

Pada umumnya solusi akan berbentuk :

$$f(i,S) = \underset{j \in S}{\text{or}}\{f(j, S-j)\ op\ j\} \qquad (3)$$

dengan or adalah fungsi yang mengambil nilai operasi boolean or untuk semua klausa dan op adalah operator yang menghubungkan fungsi rekursif dan klausa dengan variabel j.

B. Maximum Interval

Pada beberapa kasus pemilihan obyek, konfigurasi pemilihan yang digunakan tidak penting. Yang penting adalah bahwa pengambilan mungkin dilakukan dalam rentang interval yang telah dicapai sekarang.

Misal, pada masalah *longest increasing subsequence* (LIS). Masalah LIS yaitu pencarian subsekuens terpanjang pada sebuah list bilangan real. Perhatikan bahwa kata subsekuens berarti tidak selalu berurutan.

Dengan *bruteforce* dan *backtrack*, solusi dicari dengan mengenumerasi semua kemungkinan pemilihan angka yang akan dimasukkan dalam list. Sehingga kompleksitas waktu total $O(2^n)$. Kita lihat bahwa menggunakan program dinamis, kompleksitas waktu dapat direduksi hingga $O(n^2)$.

Misal kita memiliki sekuens S dengan panjang N. Lalu kita bergerak dari 1 hingga N untuk mencari LIS dari list[1…i] dengan i adalah indeks posisi kita sekarang. Kita bisa memperpanjang LIS dengan menambahkan list dengan angka pada posisi [i+1…N] **yang lebih besar dari S[i].** Atau juga bisa dibalik, untuk posisi i kita hanya perlu menambahkan S[i] ke LIS[k] dengan **S[k] < S[i] dan k < i**.

Solusi rekurens dari LIS dengan program dinamis :

$$LIS(i) = \max_{j<i\ and\ S[j]<S[i]}\{LIS(j)+1\} \quad (4)$$

Kompleksitas waktu total LIS dengan program dinamis yaitu $O(n^2)$.

Berikut adalah contoh uji kasus LIS dengan program dinamis. Kasus yang akan diuji = {-7, 10, 9, 2, 3, 8, 8, 1}.

Tahap	LIS							
	1	2	3	4	5	6	7	8
1	1	1	1	1	1	1	1	1
2	1	2	1	1	1	1	1	1
3	1	2	2	1	1	1	1	1
4	1	2	2	2	1	1	1	1
5	1	2	2	2	3	1	1	1
6	1	2	2	2	3	4	1	1
7	1	2	2	2	3	4	4	1
8	1	2	2	2	3	4	4	2

Solusinya adalah max(LIS[i]) = 4, didapat dari -7, 2, 3, 8.

C. Constraint Bounded

Pada skala praktis, bisa jadi resources yang kita miliki terbatas. Sehingga dalam optimasi ada batasan lain yang harus dipenuhi dalam pencarian solusi. Pengecekan batasan mungkin bisa jadi menambah kompleksitas program. Tapi jika pembatasan itu digunakan untuk pencarian itu sendiri dan tidak perlu mencari di luar batas, selain pengecekan tidak diperlukan, kinerja program dapat diefisienkan.

Kita ambil contoh pada persoalan Knapsack 0-1. Persoalan ini bisa diselesaikan dengan cara *set coverage* seperti yang dibahas sebelumnya. Tapi pada kenyataannya, barang yang bisa diangkut tidak tak terbatas dan biasanya dalam jangkauan yang cukup kecil. Sehingga kita bisa gunakan itu sebagai parameter fungsi PD. Maka kita bisa gunakan jangkauan total berat bawaan menjadi status fungsi.

Perhatikan bahwa banyak bawaan mempengaruhi keputusan untuk menambah atau tidak dari barang bawaan. Tidak hanya nilai barang saja yang dipertimbangkan. Maka, jika kita mampu membawa barang dengan bobot tertentu, maka hasil optimalnya juga bergantung pada hasil optimal sebelum mengambil barang tersebut. Lalu untuk tiap barang yang diambil dari status sekarang, cari nilai maksimum yang bisa dibawa (atau bahkan tidak perlu membawa jika perlu).

Secara formal, fungsi rekursifnya dapat ditulis :

$$dp(n,W) = \max\{dp(n-1,W), dp(n-1, W-w_n)+p_n\} \qquad (5)$$

Kompleksitas waktu total program dinamis ini $O(nW)$. Untuk nilai *W* yang terbatas, kompleksitas algoritma ini lebih efisien daripada $O(n2^n)$ dengan *exhaustive search*. Dan untuk kasus nyata, W hampir selalu kurang dari 2^n.

IV. KESIMPULAN

Beberapa pendekatan yang digunakan di sini adalah sebagian kecil dari teknik yang umum digunakan dalam PD. Ada teknik yang menggunakan struktur data tree dalam penyimpanan memo sehingga kompleksitas ruang bisa lebih dihemat.

Optimasi yang lain bisa dilakukan dengan mempercepat proses komputasi dengan model geometri (*convex hull*). Tapi itu diluar bahasan di kuliah ini.

TINJAUAN PUSTAKA

[1] Garey, M. R.; Johnson, D. S.; Victor Klee, ed. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. A Series of Books in the Mathematical Sciences. San Francisco, California: W. H. Freeman and Co. pp. x+338, 1979.

[2] Tarski, A.; Mostovski, A.; Robinson, R. , *Undecidable Theories*, Studies in Logic and the Foundation of Mathematics, North-Holland, Amsterdam, 1953.

[3] Bellman, Richard. *Dynamic Programming*, Princeton University Press. Dover paperback edition, 2003

[4] K. H. Rosen. Discrete Mathematics and Its Applications 7th. New York: McGraw-Hil, 2012

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 Mei 2015

Afrizal Fikri