

Bab 2

Teknik Pencarian

Bab ini membahas bagaimana membuat ruang masalah untuk suatu masalah tertentu. Sebagian masalah mempunyai ruang masalah yang dapat diprediksi, sebagian lainnya tidak.

1.1 Pendefinisian Masalah Sebagai Pencarian Ruang Keadaan

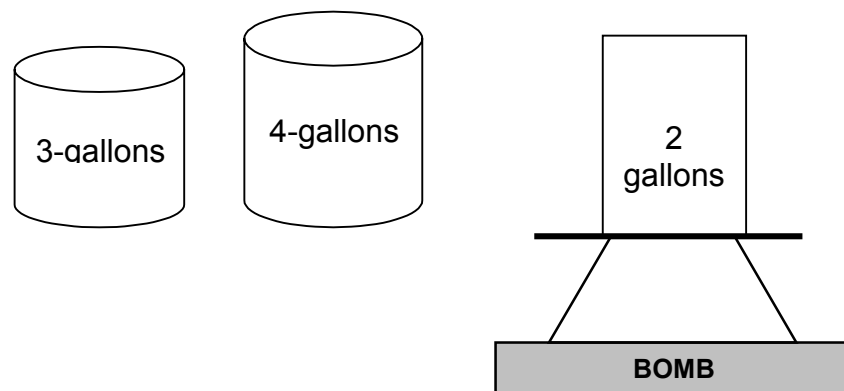
Masalah utama dalam membangun sistem berbasis AI adalah bagaimana mengkonversikan situasi yang diberikan ke dalam situasi lain yang diinginkan menggunakan sekumpulan operasi tertentu.

A Water Jug Problem

Anda diberi dua buah gelas, yang satu ukuran 4 galon dan yang lain 3 galon. Kedua gelas tidak memiliki skala ukuran. Terdapat pompa yang dapat digunakan untuk mengisi gelas dengan air. Bagaimana anda mendapatkan tepat 2 galon air di dalam gelas 4 ukuran galon?

Ruang masalah untuk masalah di atas dapat digambarkan sebagai himpunan pasangan bilangan bulat (x,y) yang terurut, sedemikian hingga $x = 0, 1, 2, 3,$ atau 4 dan $y = 0, 1, 2,$ atau 3 ; x menyatakan jumlah air dalam gelas ukuran 4 galon, dan y menyatakan jumlah air dalam gelas ukuran 3 galon. Keadaan mula-mula adalah $(0,0)$. *State* tujuan adalah $(2,n)$ untuk setiap nilai n .

Operator-operator (aturan produksi) yang digunakan untuk memecahkan masalah terlihat pada gambar 2.1.



Gambar 2.1 Suatu masalah: *Water jug Problem*.

1. (x,y)
If $x < 4$ → $(4,y)$ Isi penuh gelas 4 galon
2. (x,y)
If $y < 3$ → $(x,3)$ Isi penuh gelas 3 galon
3. (x,y)
If $x > 0$ → $(x-d,y)$ Buang sebagian air dari gelas 4 galon
4. (x,y)
If $y > 0$ → $(x,y-d)$ Buang sebagian air dari galon ukuran 3 galon
5. (x,y)
If $x > 0$ → $(0,y)$ Kosongkan gelas 4 galon
6. (x,y)
If $y > 0$ → $(x,0)$ Kosongkan gelas 3 galon
7. (x,y)
If $x+y \geq 4$ and $y > 0$ → $(4,y-(4-x))$ Tuangkan air dari gelas 3 galon ke gelas 4 galon sampai gelas 4 galon penuh
8. (x,y)
If $x+y \geq 3$ and $x > 0$ → $(x-(3-y),3)$ Tuangkan air dari gelas 4 galon ke gelas 3 galon sampai gelas 3 galon penuh
9. (x,y)
If $x+y \leq 4$ and $y > 0$ → $(x+y,0)$ Tuangkan seluruh air dari gelas 3 galon ke gelas 4 galon
10. (x,y)
If $x+y \leq 3$ and $x > 0$ → $(0,x+y)$ Tuangkan seluruh air dari gelas 4 galon ke gelas 3 galon
11. $(0,2)$ → $(2,0)$ Tuangkan 2 galon air dari gelas 3 galon ke gelas 4 galon
12. $(2,y)$ → $(0,y)$ Buang 2 galon dalam gelas 4 galon sampai habis.

Gambar 2.2 Aturan produksi untuk *Water Jug Problem*.

Jumlah galon dalam gelas 4 galon	Jumlah galon dalam gelas 3 galon	Aturan yang dilakukan
0	0	-
0	3	2
3	0	9
3	3	2
4	2	7
0	2	5 atau 12
2	0	9 atau 11

Gambar 2.3 Suatu solusi untuk *Water Jug Problem*.

1.2 Sistem Produksi

Sistem produksi terdiri dari:

- **Himpunan aturan**, masing-masing terdiri dari sisi kiri (pola) yang menentukan kemampuan aplikasi dari aturan tersebut dan sisi kanan yang menggambarkan operasi yang dilalukan jika aturan dilaksanakan.
- Satu atau lebih **pengetahuan** atau basis data yang berisi informasi apapun untuk tugas tertentu. Beberapa bagian basis data bisa permanen, dan bagian yang lain bisa hanya merupakan solusi untuk masalah saat ini. Informasi dalam basis data ini disusun secara tepat.
- **Strategi kontrol** yang menspesifikasikan urutan dimana aturan akan dibandingkan dengan basis data dan menspesifikasikan cara pemecahan masalah yang timbul ketika beberapa aturan sesuai sekaligus pada waktu yang sama.
- **A rule applier** (*pengaplikasi aturan*).

2.2.1 Strategi Kontrol

Syarat-syarat strategi kontrol:

- **cause motion**
Perhatikan kembali *water jug problem*. Jika kita mengimplementasikan strategi kontrol sederhana dengan selalu memilih aturan pertama pada daftar 12 aturan yang telah dibuat, maka kita tidak akan pernah memecahkan masalah. Strategi kontrol yang tidak menyebabkan **motion** tidak akan pernah mencapai solusi.
- **systematic**
Strategi kontrol sederhana yang lain untuk *water jug problem*: pada setiap siklus, pilih secara random aturan-aturan yang dapat diaplikasikan. Strategi ini lebih baik dari yang pertama, karena menyebabkan **motion**. Pada akhirnya strategi tersebut akan mencapai solusi. Tetapi mungkin kita akan mengunjungi beberapa *state* yang sama selama proses tersebut dan mungkin menggunakan lebih banyak langkah dari jumlah langkah yang diperlukan. Hal ini disebabkan strategi kontrol tersebut tidak sistematis. Beberapa strategi kontrol yang sistematis telah diusulkan, yang biasa disebut sebagai metoda-metoda dalam teknik *searching*. Di bab ini, akan dibahas enam metoda, yaitu *Breadth First Search*, *Uniform Cost Search*, *Depth First Search*, *Depth-Limited Search*, *Iterative-Deepening Depth-First Search*, dan *Bi-directional search*. Masing-masing metoda tersebut mempunyai karakteristik yang berbeda.

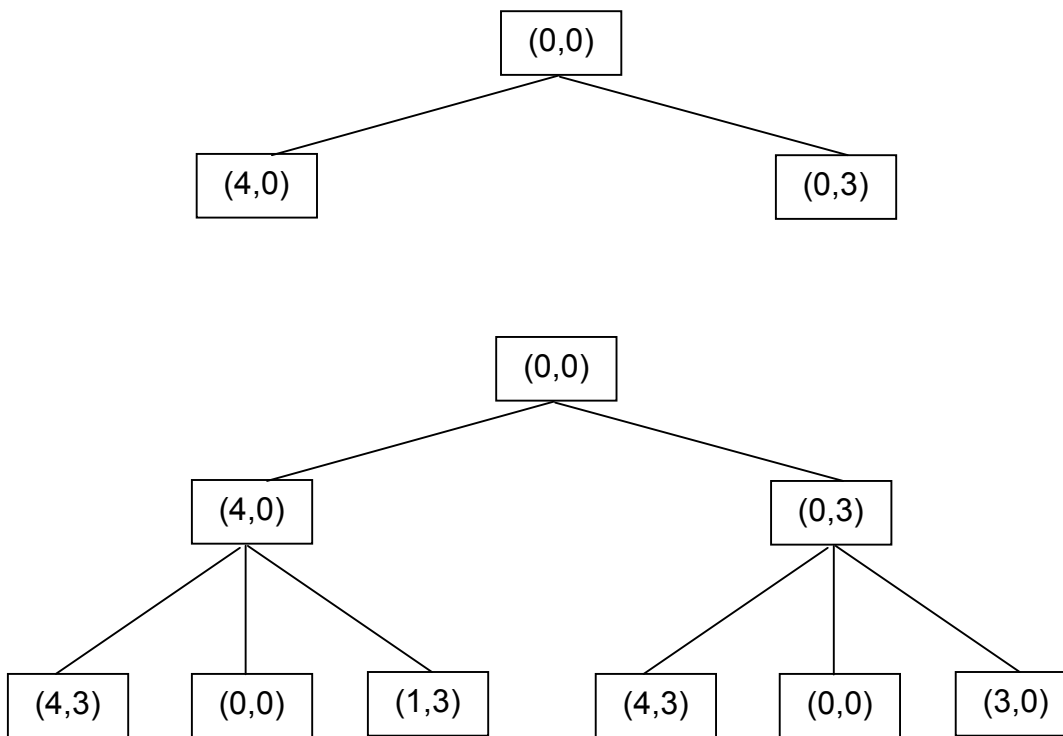
2.2.2 Strategi Pencarian

Terdapat empat kriteria dalam strategi pencarian, yaitu:

- **Completeness**: Apakah strategi tersebut menjamin penemuan solusi jika solusinya memang ada?
- **Time complexity**: Berapa lama waktu yang diperlukan?
- **Space complexity**: Berapa banyak memori yang diperlukan?
- **Optimality**: Apakah strategi tersebut menemukan solusi yang paling baik jika terdapat beberapa solusi berbeda pada permasalahan yang ada?

1. Breadth-First Search (BFS)

Pencarian dilakukan pada semua node dalam setiap level secara berurutan dari kiri ke kanan. Jika pada satu level belum ditemukan solusi, maka pencarian dilanjutkan pada level berikutnya. Demikian seterusnya sampai ditemukan solusi. Dengan strategi ini, maka dapat dijamin bahwa solusi yang ditemukan adalah yang paling baik (*Optimal*). Tetapi BFS harus menyimpan semua node yang pernah dibangkitkan. Hal ini harus dilakukan untuk penelusuran balik jika solusi sudah ditemukan. Gambar 2.4 mengilustrasikan pembangkitan pohon BFS untuk masalah *Water Jug*. Pembangkitan suksesor dari suatu node bergantung pada urutan dari Aturan Produksi yang dibuat (lihat gambar 2.2). Jika urutan dari aturan 4 ditukar dengan aturan 5, maka pohon BFS yang dibangkitkan juga akan berubah.



Gambar 2.4 Pohon *Breadth First Search* untuk *Water Jug Problem*.

2. Uniform Cost Search (UCS)

Konsepnya hampir sama dengan BFS, bedanya adalah bahwa BFS menggunakan urutan level dari yang paling rendah sampai yang paling tinggi. Sedangkan UCS menggunakan harga terendah yang dihitung berdasarkan harga dari node asal menuju ke node tersebut atau biasa dilambangkan sebagai $g(n)$. BFS adalah juga UCS jika harga $g(n) = \text{DEPTH}(n)$.

Syarat yang harus dipenuhi oleh pohon UCS: $g(\text{SUCCESSOR}(n)) \geq g(n)$ untuk setiap node n . Jika syarat ini tidak dipenuhi maka UCS tidak bisa dipakai.

3. Depth-First Search (DFS)

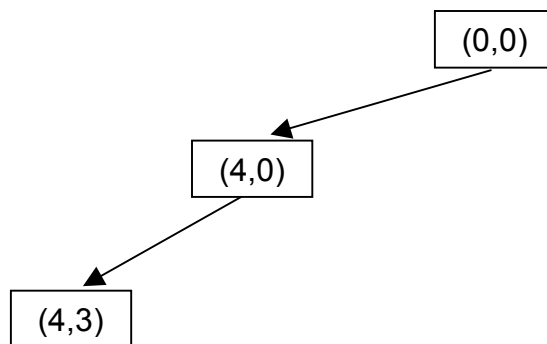
Pencarian dilakukan pada satu node dalam setiap level dari yang paling kiri. Jika pada level yang paling dalam, solusi belum ditemukan, maka pencarian dilanjutkan pada node sebelah kanan. Node yang kiri dapat dihapus dari memori. Jika pada level yang paling dalam tidak ditemukan solusi, maka pencarian dilanjutkan pada level sebelumnya. Demikian seterusnya sampai ditemukan solusi. Jika solusi ditemukan maka tidak diperlukan proses *backtracking* (penelusuran balik untuk mendapatkan jalur yang diinginkan).

Kelebihan DFS adalah:

- Pemakaian memori hanya sedikit, berbeda jauh dengan BFS yang harus menyimpan semua node yang pernah dibangkitkan.
- Jika solusi yang dicari berada pada level yang dalam dan paling kiri, maka DFS akan menemukannya secara cepat.

Kelemahan DFS adalah:

- Jika pohon yang dibangkitkan mempunyai level yang dalam (tak terhingga), maka tidak ada jaminan untuk menemukan solusi (**Tidak Complete**).
- Jika terdapat lebih dari satu solusi yang sama tetapi berada pada level yang berbeda, maka pada DFS tidak ada jaminan untuk menemukan solusi yang paling baik (**Tidak Optimal**).



Gambar 2.5 Penelusuran *Depth First Search* untuk *Water Jug Problem*.

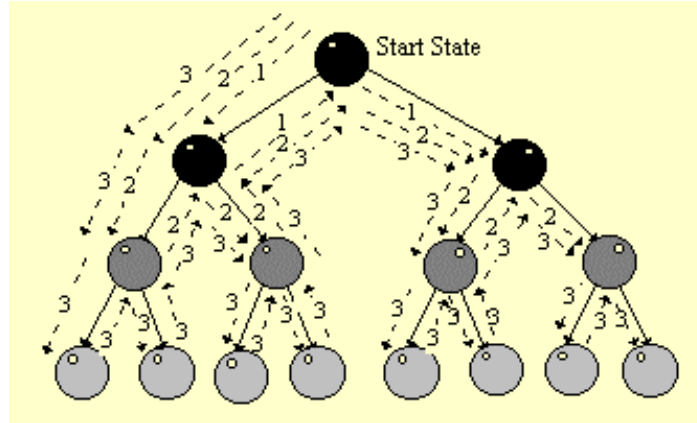
4. Depth-Limited Search (DLS)

Mengatasi kelemahan DFS (**tidak complete**) dengan membatasi kedalaman maksimum dari suatu jalur solusi. Tetapi harus diketahui atau ada batasan dari sistem tentang level maksimum. Jika batasan kedalaman terlalu kecil, DLS tidak *complete*.

5. Iterative-Deepening Depth-First Search (IDS)

Merupakan metode yang berusaha menggabungkan keuntungan BFS (**Complete** dan **Optimal**) dengan keuntungan DFS (**Space Complexity** yang rendah). Tetapi konsekuensinya adalah Time Complexity-nya menjadi tinggi. Perhatikan gambar 2.6. Pencarian dilakukan secara iteratif (menggunakan penelusuran DFS) dimulai dari

batasan level 1. Jika belum ditemukan solusi, maka dilakukan iterasi ke-2 dengan batasan level 2. Demikian seterusnya sampai ditemukan solusi. Jika solusi ditemukan maka tidak diperlukan proses *backtracking* (penelusuran balik untuk mendapatkan jalur yang diinginkan).

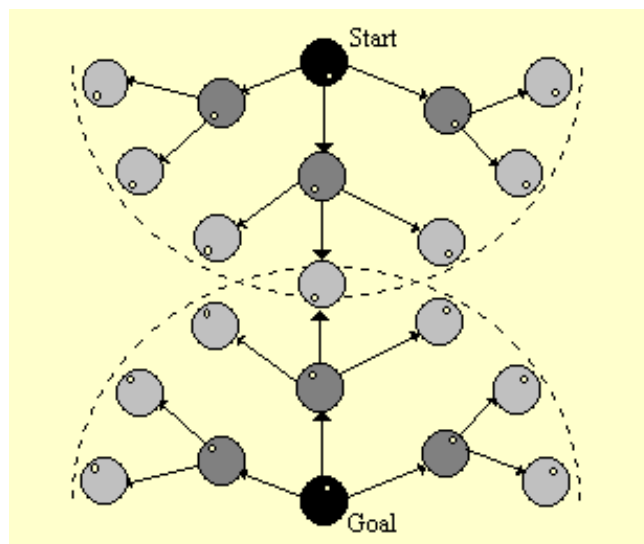


Gambar 2.6 Penelusuran *Iterative Deepening DFS*.

6. Bi-Directional Search

Pada setiap iterasi, pencarian dilakukan dari dua arah: dari node asal (*start*) dan dari node tujuan (*goal*). Ketika dua arah pencarian membangkitkan node yang sama, maka solusi telah ditemukan, dengan cara menggabungkan kedua jalur yang bertemu. Ada beberapa masalah sebelum memutuskan untuk menggunakan strategi *Bi-directional Search*, yaitu:

- Bagaimana kalau terdapat beberapa node tujuan yang berbeda?
- Terdapat perhitungan yang tidak efisien untuk selalu mengecek apakah node baru yang dibangkitkan sudah pernah dibangkitkan oleh pencarian dari arah yang berlawanan.
- Bagaimana menentukan strategi pencarian untuk kedua arah tersebut? Misalnya dari arah sumber dan dari arah tujuan digunakan BFS.



Gambar 2.7 Penelusuran *Bi-directional Search*.
Perbandingan Strategi Pencarian [RUS95]

Criterion	Breadth First	Uniform Cost	Depth-First	Depth-Limited	Iterative Deepening	Bidirectional (if applicable)
Time	b^d	b^d	b^m	b^l	b^d	$b^{d/2}$
Space	b^d	b^d	bm	bl	bd	$b^{d/2}$
Optimal?	Yes	Yes	No	No	Yes	Yes
Complete?	Yes	Yes	No	Yes, if $l \geq d$	Yes	Yes

Keterangan:

b : faktor pencabangan (the branching factor)

d : kedalaman solusi (the depth of solution)

m : kedalaman maksimum pohon pencarian (the maximum depth of the search tree)

l : batasan kedalaman (the depth limit)