Heuristic Search

(atau Intelligent Search, Rule of Thumb Search, Informed Search)

Problem pada Blind Search (Brute Force Search, Systematic Search, Uninformed Search): *Combinatorial Explosion*. Inilah sebabnya blind search sering disebut sebagai Exhaustive Search (Pencarian Solusi yang Melelahkan).

Pada heuristic search, ditambahkan informasi khusus yang spesifik untuk masalahnya (domain-specific information) saat memilih path / operator yang dianggap terbaik dalam melanjutkan pencarian solusi.

Didefinisikan sebuah fungsi heuristic, h(n), yang mengestimasi *"seberapa baiknya"* pilihan pada state n. Fungsi ini akan menentukan kualitas setiap state dalam state-space.

h(n) yang baik haruslah cepat untuk dihitung.

Karakteristik nilai h(n):

- h(n)>=0, untuk semua state n
- h(n)=0, menunjukkan bahwa n adalah sebuah goal state
- h(n)=tak terbatas, menunjukkan bahwa state n adalah sebuah kondisi saat mana sebuah goal state tidak mungkin terjangkau.

Hasil dari heuristic search boleh jadi "kurang optimal", tetapi dapat dikatakan "cukup baik" selama fungsi heuristik yang digunakan juga "cukup cerdas".

Keuntungan penerapan heuristic search dalam pencarian solusi:

- Heuristic search memiliki fleksibilitas tinggi yang memungkinkan untuk digunakan pada masalah yang kompleks dan tidak terstruktur.
- Blind search menjamin ditemukannya solusi optimal, tetapi kurang layak untuk digunakan dalam komputasi karena kebutuhan memory (dan waktu) yang sangat besar.
- Metode heuristic lebih sederhana untuk dipahami oleh pengambil keputusan, secara khusus apabila didukung oleh *analisis kualitatif*.
- Suatu metode heuristic dapat digunakan sebagai bagian dari prosedur iteratif yang tetap menjamin ditemukannya sebuah solusi optimal.

Road Map (Route Finding)

h(x) : Euclidian distance (akar dari dx^2 + dy^2) antara 2 kota yang berbeda
Nilai *minimal* adalah yang dianggap terbaik.

Missionaries and Cannibals

Jumlah orang (total misionari dan kanibal) di tepi kiri sungai.

8-Puzzle

h1(x) : Jumlah kotak yang tidak cocok (mismatched) pada 2 konfigurasi papan yang berbeda

h2(x) : Jumlah dari jarak vertikal dan horizontal dari kotak-kotak yang tidak cocok (mismatched) pada2 konfigurasi papan yang berbeda, dikenal dengan nama Manhattan (City-Block) distance.

h3(x) : 3 * seq(k), dimana seq(k)=1 untuk kotak (bukan kosong) yang di tengah, dan seq(n)=2 untuk setiap kotak (yang terletak pada tepian) yang tidak diikuti *proper successor*nya. Proper successor dari kotak i adalah kotak yang harus mengikutinya pada konfigurasi goal state.

h4(x) dari Nils Nilsson : seq(k), dimana seq(n)=2 untuk setiap kotak (yang terletak pada tepian) yang tidak diikuti *proper successor*nya, seq(n)=1 untuk kotak (bukan kosong) yang di tengah, dan seq(n)=0 untuk kotak lainnya (berarti kotak yang diikuti proper succesornya).

Untuk sebuah *difficult / hard problem* 8-puzzle dengan konfigurasi berikut:

```
Start :  216   Goal :  123
         4-8           8-4
         753           765
```

Pemakaian h4(x) dari Nils Nilsson akan membutuhkan 18 moves (sequence of operators) dengan search tree yang *hanya* berjumlah 44 nodes.

Nilai *minimal* adalah yang terbaik.

Contoh:

```
Start :  2-3  Goal :   123
         184           8-4
         765           765

left :   -23  down :   283   right : 23-
         184           1-4           184
         765           765           765
```

dengan h1(x):

- h(left)=2 untuk kotak 1 dan 8
- h(down)=3 untuk kotak 1,2, dan 8
- h(right)=4 untuk kotak 1,2,3 dan 8

dengan h2(x):

- h(left)=2 untuk kotak 1 (0+1=1) dan 8 (1+0=1)
- h(down)=3 untuk kotak 1 (0+1=1), 2 (1+0=1), dan 8 (1+0)
- h(right)=4 untuk kotak 1 (0+1=1), 2 (1+0=1), 3 (1+0=1), dan 8 (0+1=1)

Eight Queens

h1(x): Jumlah sel yang belum terancam oleh queen
h2(x): Jumlah sel minimum yang belum terancam pada setiap barisnya
Awas: Nilai *maximal* adalah yang terbaik. Ini berbeda dengan sifat h(x) yang biasa digunakan.
Contoh:

```
Pada level 3 (ini):
---Q----
-Q------
-------Q
--A-BC-- {A,B,C adalah operator-operator yang akan dipilih}
--------
--------
--------
```

```
--------

Pada level 4 (berikutnya):
---1----
-2111---
2121-1-3
12A1--13 {Bila A dipakai}
-2A123-1 (=2)
A2A1A2-3 (=1), nilai minimum, jadi h2(A)=1
-2A1-A23 (=2)
-2A1--A2 (=3)
         ---- +
    h1(A)= 8
=====================================================================
---1----
-2111---
2221-1-3
12-1B-13 {Bila B dipakai}
-2-123-1 (=3)
-2B132B3 (=1), nilai minimum, jadi h2(B)=1
-2-1B-2B (=3)
B231B--2 (=2)
         ---- +
    h1(B)= 9
=====================================================================
---1----
-2111---
-221-1-3
12-1-C13 {Bila C dipakai}
-2-12CC1 (=2), nilai minimum
-2-132-3 (=3)
-2C1-C23 (=2), atau baris ini juga nilai minimum, jadi
h2(C)=2
-231-C-3 (=3)
         ---- +
    h1(C)=10
```

Pengukuran Unjuk kerja dari State-Space Solution Search

Dua cara populer untuk mengukur unjuk kerja dari State-Space Solution Search adalah *penetrance* dan effective *branching factor*.

Penetrance dihitung secara sederhana dengan rumus:

$$P = \frac{L}{N}$$

dimana: L adalah panjang dari path solusi, dan
N adalah total jumlah node yang digenerate dalam state-space

Secara umum penetrance yang lebih besar menunjukkan performansi algoritma yang lebih baik. Branching Factor adalah rata-rata children yang diexpand untuk setiap node dalam tree. Search yang dilakukan dengan menggunakan heuristic sempurna, h(x)=h'(x) akan menyebabkan branching factor=1. Artinya algoritma hanya expand sebuah child yang selalu merupakan optimal search path.