Teknik Pencarian Heuristik

Generate and Test
Hill Climbing
Best First Search
Problem Reduction
Constraint Satisfaction
Means End Analysis

Referensi
Sri Kusumadewi - bab 2
Rich & Knight – bab 3

Pencarian Heuristik (Heuristic Search)

- Heuristik adalah sebuah teknik yang mengembangkan efisiensi dalam proses pencarian, namum dengan kemungkinan mengorbankan kelengkapan (*completeness*).

- Fungsi heuristik digunakan untuk mengevaluasi keadaan-keadaan problema individual dan menentukan seberapa jauh hal tersebut dapat digunakan untuk mendapatkan solusi yang diinginkan.

- Jenis-jenis *Heuristic Searching*:
  - *Generate and Test.*
  - *Hill Climbing.*
  - *Best First Search.*
  - *Alpha Beta Prunning,Means-End-Anlysis,Constraint Satisfaction, Simulated Annealing,* dll

PEMBANGKITAN dan PENGUJIAN (*Generate and Test*)

Metode ini merupakan penggabungan antara *depth-first search* dengan pelacakan mundur (*backtracking*), yaitu bergerak ke belakang menuju pada suatu keadaan awal.

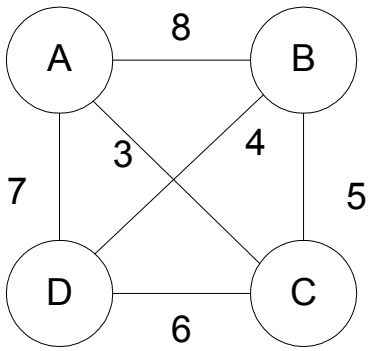
Algoritma :

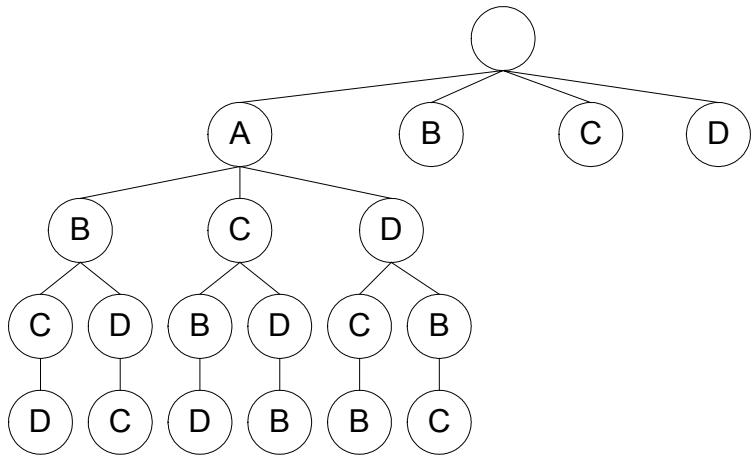
1. Bangkitkan suatu kemungkinan solusi (membangkitkan suatu tititk tertentu atau lintasan tertentu dari keadaan awal).
2. Uji untuk melihat apakah node tersebut benar-benar merupakan solusinya dengan cara membandingkan node terebut atau node akhir dari suatu lintasan yang dipilih dengan kumpulan tujuan yang diharapkan.
3. Jika solusi ditemukan, keluar. Jika tidak, ulangi kembali langkah pertama.

Contoh : **"Travelling Salesman Problem (TSP)"**

- Seorang salesman ingin mengunjungi n kota. Jarak antara tiap-tiap kota sudah diketahui. Kita ingin mengetahui ruter terpendek dimana setaip kota hanya boleh dikkunjungi tepat 1 kali. Misalkan ada 4 kota dengan jarak antara tiap-tiap kota seperti berikut ini :

Penyelesaian dengan metode *Generate and Test*





Alur pencarian dengan *Generate and Test*

Pencarian ke-	Lintasan	Panjang Lintasan	Lintasan Terpilih	Panjang Lintasan Terpilih
1	ABCD	19	ABCD	19
2	ABDC	18	ABDC	18
3	ACBD	12	ACBD	12
4	ACDB	13	ACBD	12
5	ADBC	16	ACBD	12
Dst…				

PENDAKIAN BUKIT (*Hill Climbing*)

Metode ini hampir sama dengan metode pembangkitan dan pengujian, hanya saja proses pengujian dilakukan dengan menggunakan fungsi heuristic. Pembangkitan keadaan berikutnya tergantung pada feedback dari prosedur pengetesan. Tes yang berupa fungsi heuristic ini akan menunjukkan seberapa baiknya nilai terkaan yang diambil terhadap keadaan-keadaan lainnyayang mungkin.

Algoritma *Simple Hill Climbing*

1. Cari operator yang belum pernah digunakan; gunakan operator ini untuk mendapatkan keadaan yang baru.

a) Kerjakan langkah-langkah berikut sampai solusinya ditemukan atau sampai tidak ada operator baru yang akan diaplikasikan pada keadaan sekarang :
Cari operator yang belum digunakan; gunakan operator ini untuk mendapatkan keadaan yang baru.

b) Evaluasi keadaan baru tersebut :
   - Jika keadaan baru merupakan tujuan, keluar
   - Jika bukan tujuan, namun nilainya lebih baik daripada keadaan sekarang, maka jadikan keadaan baru tersebut menjadi keadaan sekarang.
   - Jika keadaan baru tidak lebih baik daripada keadaan sekarang, maka lanjutkan iterasi.

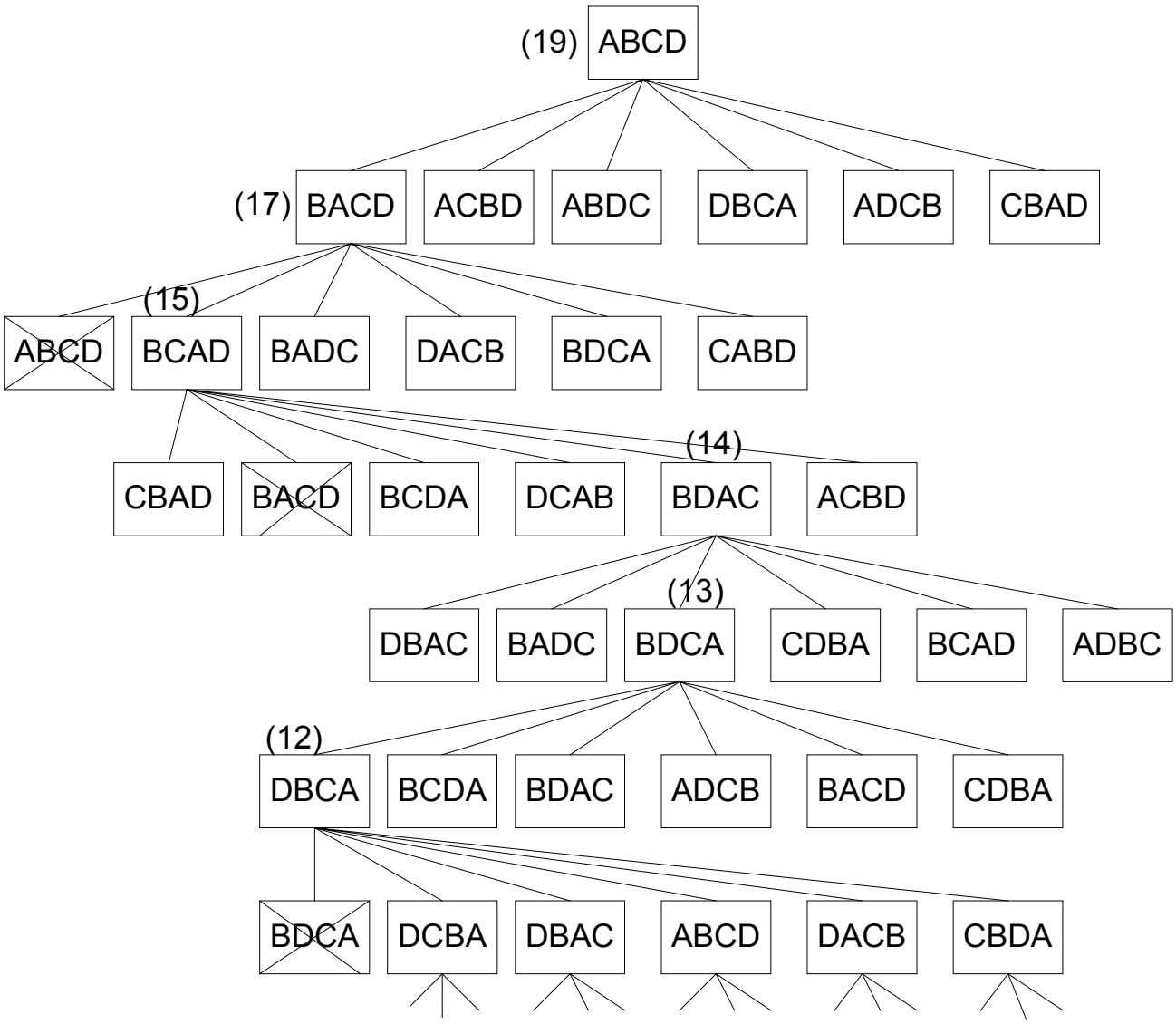
Pada *simple hill climbing*, ada 3 masalah yang mungkin:

- Algoritma akan berhenti kalau mencapai nilai optimum local
- Urutan penggunaan operator akan sangat berpengaruh pada penemuan solusi
- Tidak diijinkan untuk melihat satupun langkah sebelumnya.

Contoh : TSP dengan *Simple Hill Climbing*

Disini ruang keadaan berisi semua kemungkinan lintasan yang mungkin. Operator digunakan untuk menukar posisi kota-kota yang bersebelahan. Apabila ada n kota, dan kita ingin mencari kombinasi lintasan dengan menukar posisi urutan 2 kota, maka kita akan mendapatkan sebanyak $\frac{n!}{2!(n-2)!}$ atau sebanyak 6 kombinasi.

Fungsi heuristic yang digunakan adalah panjang lintasan yang terjadi.



PENCARIAN TERBAIK PERTAMA (*Best-First Search*)

Metode ini merupakan kombinasi dari metode *depth-first search* dan *breadth-first search*. Pada metode *best-first search*, pencarian diperbolehkan mengunjungi node yang ada di level yang lebih rendah, jika ternyata node pada level yang lebih tinggi ternyata memiliki nilai heuristic yang lebih buruk.

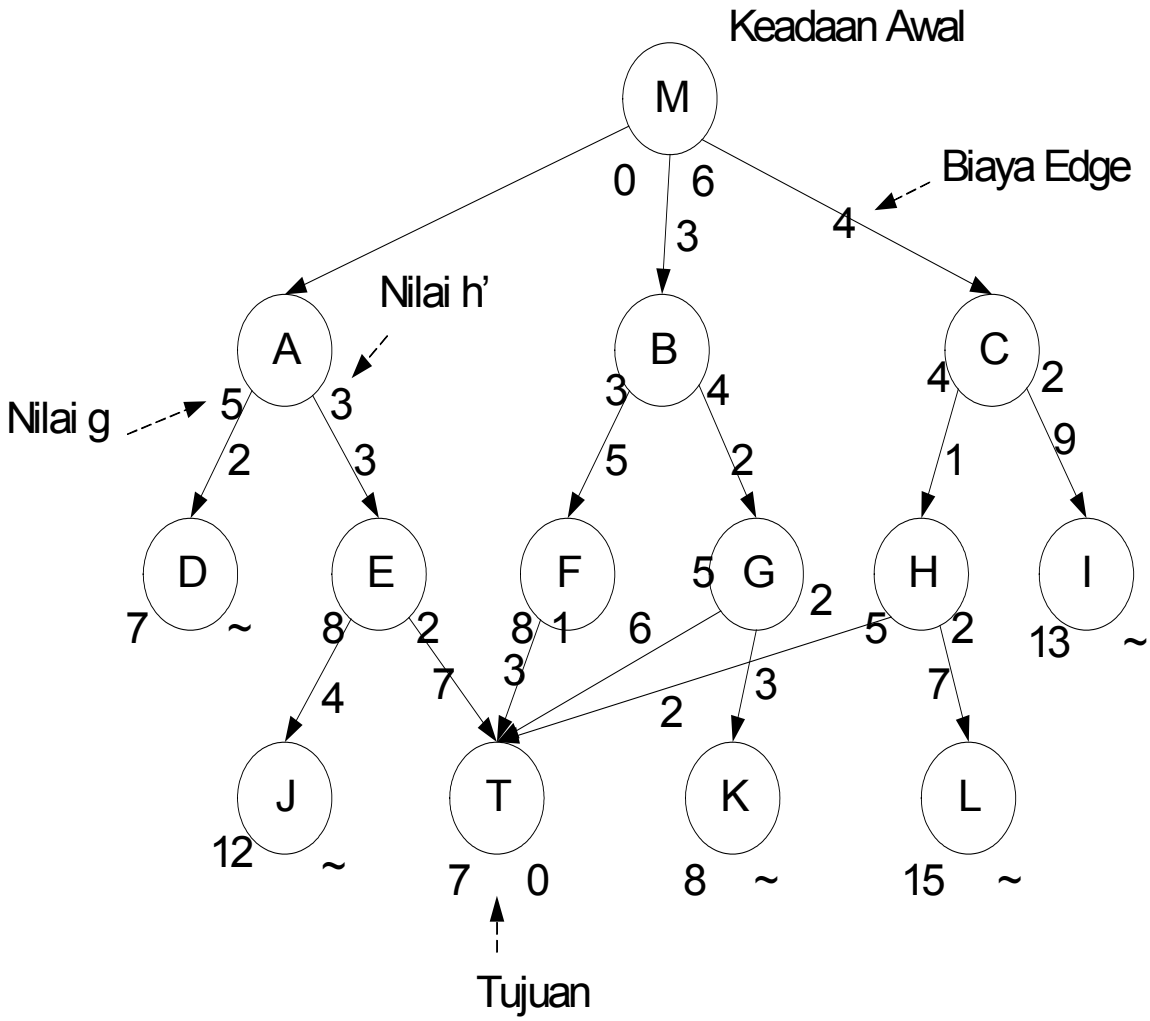
Fungsi Heuristik yang digunakan merupakan prakiraan (estimasi) *cost* dari *initial state* ke *goal state*, yang dinyatakan dengan :

$$f'(n) = g(n) + h'(n)$$

dimana f' = Fungsi evaluasi
g = *cost* dari *initial state* ke *current state*
h' = prakiraan *cost* dari *current state* ke *goal state*

Contoh :

Misalkan kita memiliki ruang pencarian seperti pada gambar berikut. Node M merupakan keadaan awal dan node T merupakan tujuannya. Biaya edge yang menghubungkan node M dengannode A adalah biaya yang dikeluarkan untuk bergerak dari kota M ke kota A. Nilai g diperoleh berdasarkan biaya edge minimal. Sedangkan nilai h’ di node A merupakan hasil perkiraan terhadap biaya yang diperlukan dari node A untuk sampai ke tujuan. h’(n) bernilai ~ jika sudah jelas tidak ada hubungan antara node n dengan node tujuan (jalan buntu). Kita bisa merunut nilai untuk setiap node.


Keadaan Awal
M
Biaya Edge
0
6
3
4
A
B
C
Nilai h'
Nilai g
5
3
2
3
3
4
5
2
4
2
1
9
D
E
F
G
H
I
7
~
8
2
8
1
6
5
G
2
5
2
13
~
3
7
4
2
3
7
J
T
K
L
12
~
7
0
8
~
15
~
Tujuan

Tabel status tiap node

Node (n)	g(n)	h'(n)	f'(n)
M	0	6	6
A	5	3	8
B	3	4	7
C	4	2	6
D	7	~	~
E	8	2	10
F	8	1	9
G	5	2	7
H	5	2	7
I	13	~	~
J	12	~	~
K	8	~	~
L	15	~	~
T	7	0	7

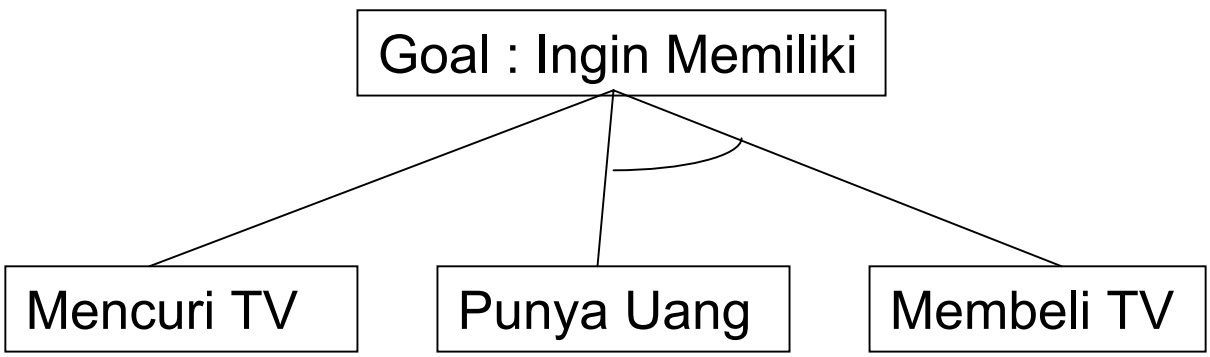
Solusi : M-C H-T

Reduksi Masalah

- Kebanyakan solusi menggunakan pohon OR, dimana lintasan dari awal sampai tujuan tidak terletak pada satu cabang.
- Bila lintasan dari keadaan awal sampai tujuan dapat terletak pada satu cabang, maka kita akan dapat menemukan tujuan lebih cepat.
- Graf AND-OR
- Graf AO*

Graf AND-OR

- Pada dasarnya sama dengan algoritma Best First Search, dengan mempertimbangkan adanya arc AND.
- Gambar berikut menunjukkan bahwa untuk mendapatkan TV orang bisa dengan cara singkat yaitu mencuri atau membeli asal mempunyai uang.
- Untuk mendeskripsikan algoritma, digunakan nilai F_UTILITY untuk biaya solusi.



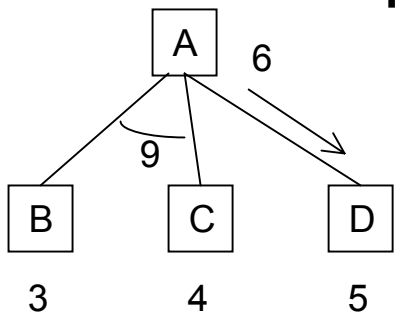
Algoritma AND-OR

1. Inisialisasi graf ke node awal.
2. Kerjakan langkah2 berikut hingga node awal SOLVED atau sampai biayanya lebih tinggi dari F_UTILITY :
   a) Telusuri graf mulai dari node awal dan ikuti jalur terbaik. Akumulasikan kumpulan node yang ada pada lintasan tsb. dan belum pernah diekspansi atau diberi label SOLVED.
   b) Ambil satu node dan ekspansi node tsb. Jika tidak ada successor maka set F_UTILITY sebagai nilai dari node tsb. Bila tidak demikian, tambahkan successor dari node tsb ke graf dan hitung nilai setiap f' (hanya gunakan h' dan abaikan g). Jika f'=0 tandai node tsb dengan SOLVED.
   c) Ubah f' harapan dari node baru yang diekspansi. Kirimkan perubahan ini secara backward sepanjang graf. Jika node berisi suatu arc successor yang semua descendant nya berlabel SOLVED maka tandai node itu dengan SOLVED.

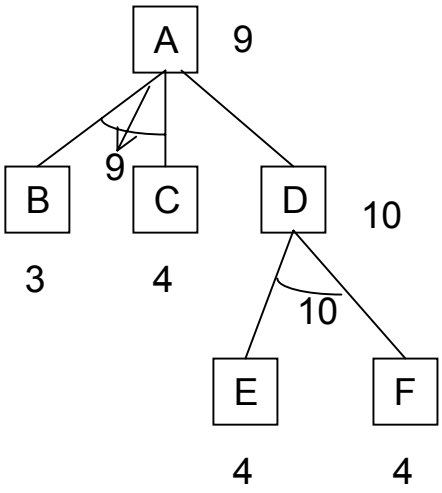
Operasi reduksi masalah dengan graf AND-OR

Sebelum step-1

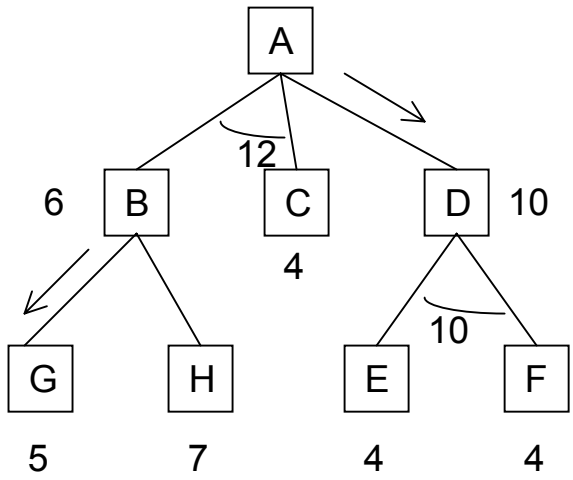
sebelum step 2



Sebelum step-3



sebelum step-4



- Langkah-1 semula hanya ada satu node, A. Node A diekspansi menjadi node B, C dan D. Node D memiliki biaya yang lebih rendah (6) jika dibandingkan dengan B dan C (9).
- Langkah-2 mengekspansi node D menjadi E dan F dengan biaya estimasi sebesar 10. f' dari D menjadi 10.
- Ternyata level sebelumnya, node B dan C memiliki biaya yang lebih rendah dari D (9 < 10).
- Pada langkah-3 telusuri arc dari A ke B dan C bersama2. Jika B dieksplore dahulu maka akan menurunkan G dan H. Nilai f' baru dari B adalah 6 (G=6 lebih baik dari H=8) sehingga biaya AND arc B-C menjadi 12 (6+4+2).
- Dengan demikian nilai node D kembali menjadi lebih baik (10<12). Sehingga ekspansi dilakukan kembali terhadap D. dst.

Algoritma AO*

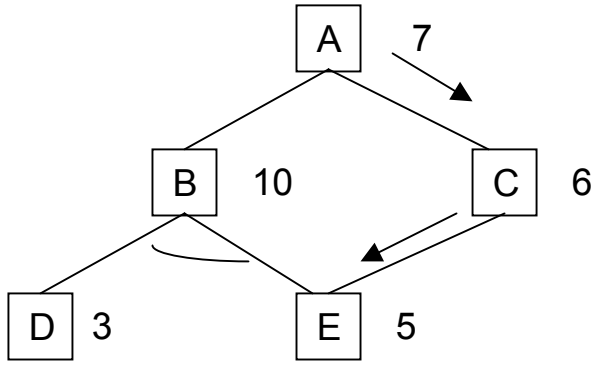
- Menggunakan struktur graf. Tiap node pada graf memiliki nilai h' yang merupakan biaya estimasi jalur dari node itu sendiri sampai suatu solusi.

- Algoritma

1. Diketahui graf yang berisi node awal (sebut saja INIT). Hitung h'(INIT)
2. Kerjakan langkah berikut hingga INIT bertanda SOLVED atau sampai nilai h'(INIT) > FUTILITY :

a) Ekspan INIT dan ambil salah satu node yang belum pernah diekspan (sebut NODE)
b) Bangkitkan successor2 NODE. Jika tidak memiliki successor maka set FUTILITY dengan nilai h'(NODE). Jika ada successor maka untuk setiap successor (sebut SUCC) yang bukan ancestor dari NODE kerjakan :
   i. Tambahkan SUCC ke graf
   ii. Jika SUCC adalah terminal node tandai dengan SOLVED dan set nilai h' = 0
   iii. Jika SUCC bukan terminal node, hitung nilai h'.
c) Kirimkan informasi baru tsb ke graf dengan cara : tetapkan S adalah node yang ditandai dengan SOLVED atau node yang nilai h'-nya baru saja diperbaiki, dan sampaikan nilai ini ke parent-nya. Inisialisasi S = NODE. Kerjakan langkah berikut ini hingga S kosong :

i. Jika mungkin, seleksi dari S node yang tidak memiliki descendant dalam graf yang terjadi pada S. Jika tidak ada, seleksi sebarang node dari S (sebut CURRENT) dan hapus dari S.
ii. Hitung biaya tiap2 arc yang muncul dari CURRENT. Biaya ini sama dengan jumlah h' untuk tiap2 node pada akhir arc ditambah dengan biaya arc itu sendiri. Set h'(CURRENT) dengan biaya minimum yang baru saja dihitung dari setiap arc yang muncul tadi.
iii. Tandai jalur terbaik yang keluar dari CURRENT dengan menandai arc yang memiliki biaya minimum.
iv. Tandai CURRENT dengan SOLVED jika semua node yang dihubungkan dengannya hingga arc yang baru saja ditandai tadi telah ditandai dengan SOLVED.
v. Jika CURRENT telah ditandai dengan SOLVED atau jika biaya CURRENT telah berubah maka status baru ini harus disampaikan ke graf. Kemudian tambahkan semua ancestor dari CURRENT ke S.



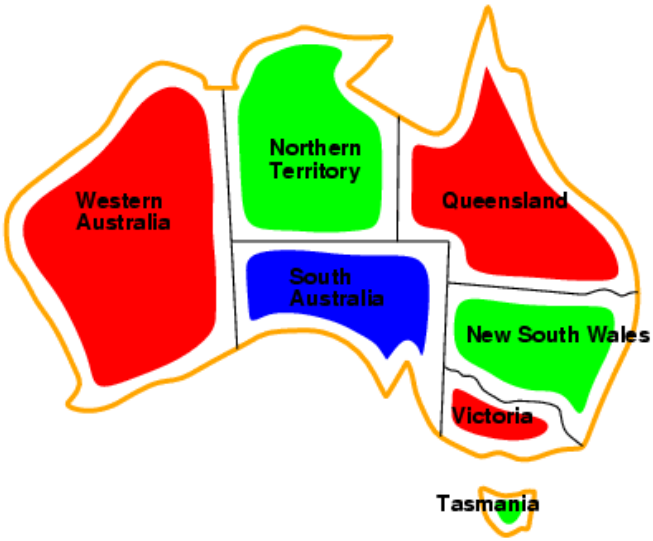
- Jalur melalui C selalu lebih baik dari B. Tetapi jika biaya node E muncul dan pengaruh perubahan yang diberikan ke node B tidak sebesar pengaruhnya terhadap node C maka jalur melalui B bisa lebih baik.
- Hasil ekspan E, misalkan 10, maka biaya node C menjadi 11 (10+1), dengan demikian biaya node A apabila melewati C adalah 12 (11+1). Tentu saja akan lebih baik memilih melalui node B (11).
- Tapi tidak demikian halnya jika kemudian node D diekspan. Bisa jadi jalur dengan melalui node B akan lebih buruk lagi ketimbang jalur yang melalui node C.

Constraint Satisfaction

- Problem search standard :
  - state adalah "black box“ – setiap struktur data yang mendukung fungsi successor, fungsi heuristik dan tes goal.
- CSP:
  - state didefinisikan sebagai variabel X_i dengan nilai dari domain D_i
  - Tes goal adalah sekumpulan constraint yang menspesifikasikan kombinasi dari nilai subset variabel.
- Contoh sederhana adalah bahasa representasi formal.

- CSP ini merupakan algoritma general-purpose dengan kekuatan lebih daripada algoritma pencarian standar.

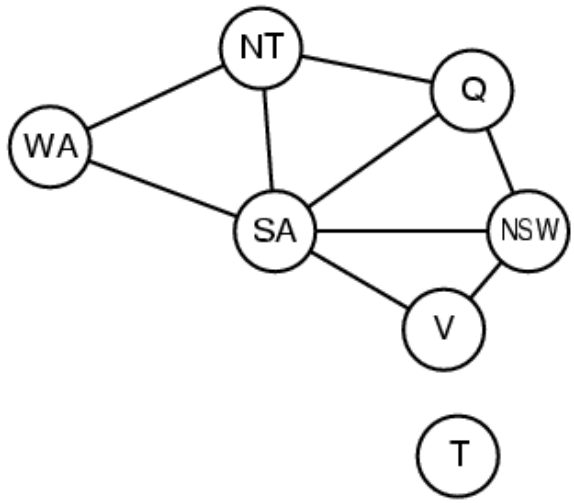
Contoh : Pewarnaan Peta



- Variabel *WA, NT, Q, NSW, V, SA, T*
- Domain D_i = {red,green,blue}
- Constraints : daerah yang bertetangga dekat harus memiliki warna yang berbeda.
- Contoh WA ≠ NT, atau (WA,NT) {(red,green),(red,blue),(green,red), (green,blue),(blue,red),(blue,green)}
- Solusi lengkap dan konsisten, contoh : WA = red, NT = green,Q = red,NSW = green,V = red,SA = blue,T = green

Constraint Graf

- Binary CSP biner : setiap constraint merelasikan dua variabel
- Graf Constraint : node adalah variabel, arc adalah constraint



MEA (Means-Ends Analysis)

- MEA adalah strategi penyelesaian masalah yang diperkenalkan pertama kali dalam GPS (General Problem Solver) [Newell & Simon, 1963].
- Proses pencarian berdasarkan ruang masalah yang menggabungkan aspek penalaran forward dan backward.
- Perbedaan antara state current dan goal digunakan untuk mengusulkan operator yang mengurangi perbedaan itu.
- Keterhubungan antara operator dan perbedaan tsb disajikan sebagai pengetahuan dalam sistem (pada GPS dikenal dengan Table of Connections) atau mungkin ditentukan sampai beberapa pemeriksaan operator jika tindakan operator dapat dipenetrasi.
- Contoh OPERATOR first-order predicate calculus dan operator2 tertentu mengijinkan perbedaan korelasi task-independent terhadap operator yang menguranginya.
- Kapan pengetahuan ada tersedia mengenai pentingnya perbedaan, perbedaan yang paling utama terpilih pertama lebih lanjut meningkatkan rata-rata capaian dari MEA di atas strategi pencarian Brute-Force.
- Bagaimanapun, bahkan tanpa pemesanan dari perbedaan menurut arti penting, MEA meningkatkan metode pencarian heuristik lain (di rata-rata kasus) dengan pemusatan pemecahan masalah pada perbedaan yang nyata antara current state dengan goal-nya.

Contoh :

- http://www.rci.rutgers.edu/~cfs/472_html/Planning/GPS_472.html
- Giarratano – hal 261-262
- Luger – hal 430 - 433.