C4 Search. Depth-first, Hill Climbing, Beam

1/3 Căutări. În adâncime, Căutarea pe deal / Urcă panta, Grinda / Raza

- o abordare intuitivă
- un model pt. lucrurile ce se întâmplă în creierul uman

Harta + S + G + taxi => exemplu + Modificare → începător; expert (thief)

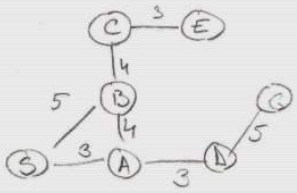
Cum funcționează: după cum urmează.

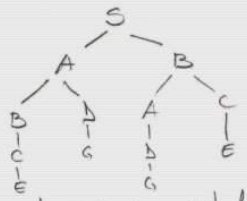
Dacă vă întreb care ar fi o cale de la S la G, oameni ar răspunde cu un trasee destul de bun dar nu optim folosind sistemul vizual (nu știe cum funcționează)

Programele noastre nu au ochi și nu au algoritmi vizuali => trebuie să facă altceva

British Museum Approach – se depistează toate căile posibile către țintă

- arborele asociat căutării se dezvoltă (abordează) în ordine alfabetică / lexical order



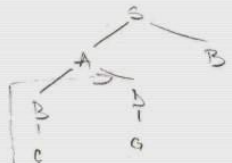


- căutarea se face „înainte"; nici un nod nu poate avea un nod fiu = nod părinte
  nod urmaș = nod înaintaș

(S → A → S – invalid)

Căutarea ≠ Hărți (Căutarea are de a face cu alegerea de opțiuni atunci când se iau decizii)
dar hărțile se folosesc drept exemple

Căutarea în adâncime (Depth-first)



→ Algoritm

fundătură => întoarcere până la prima decizie care apare pe drumul de întoarcere
BACKTRACKING

Căutarea pe nivel (Breadth-first search)
- arborele de căutare se construiește pater/nivel cu nivel până la țintă

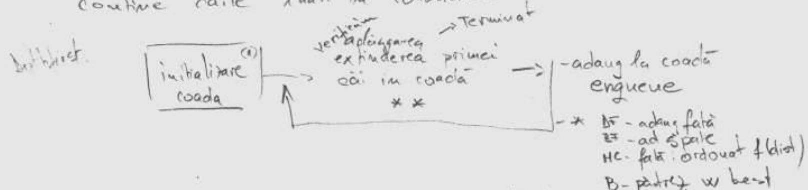
Exemplu

Căutarea pe model cu start în centru => multe căutări în stânga, nenatural pt.
(2) un om care vede că ținta e în dreapta

C4
2/3

Flowchart / Schema logică a unei căutări

- se va dezvolta o „coadă" (queue); lista de așteptare; (waiting list / live) care conține căile luate în considerare



(S) (1)
(SA) (SB)
* adăugare la stânga (DF search) începutul cozii
(SAB) (SAD) (SB)
(SABC) (SAD) (SB)
(SABCE) (SAD) (SB)
↑ nu duce la țintă și se termină => este eliminat
(SAD) (SB)
(SADG) (SB)

BF Search * adăugare se face la dreapta / la spate

Dacă nu interesează eficiența DF și BF e ok.

Optimizare. alg. sunt foarte „proști" (2) pt. că nu își poate da seama dacă se apropie sau se depărtează de țintă.

(1) - extinde noduri care au fost deja extinse
Ex: BF - extinde A de 2 ori

Concluzie. Modificăm algoritmul „puțin" -> * * extend unless final node never before extended
extinde dacă (nodul curent nu mai apare în arbore)

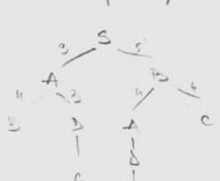
=> se va crea o listă cu noduri deja extinse - extended list

(2)
Căutarea se face și în stânga
=> necesitatea unei căutări mai „informate" - orientate
în general e un lucru bun să ne apropiem de țintă. Dintre două noduri, se alege cel mai apropiat de țintă.

=>

Hill Climbing

- algoritmul este același ca DF doar că nodurile nu sunt abordate alfabetic ci în funcție de apropierea de țintă. / distanța parcursă (ajungem cât mai repede la țintă)



- d((SBA), (G)) = d((SBC), (G)) -> abordare Lexicografică
- metoda ar fi demnă de a fi folosită dacă ∃ o metodă (euristică) de a determina distanța nodului curent față de nodul următor
- algoritm: adăugarea în listă se face în față, după care lista se ordonează în funcție de distanța față de țintă

Beam search (Căutare Fascicul)

- analog cu Breadth first search
derivat din
- se impune ca un număr limitat de noduri de pe un palier să fie extinse; ex: W = 2, cele mai aproape de țintă

- algoritm: nu contează cum se face adăugarea pt. că se vor păstra cele mai bune W soluții

L4

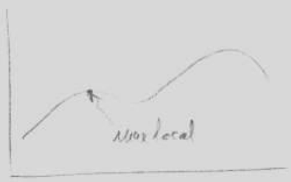
3/3 Problema căutării se pune și în spații continue. Acești algoritmi pot fi folosiți și în aceste situații.

Probleme la Hill Climbing

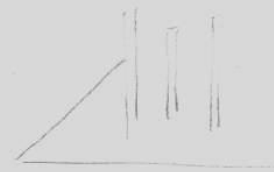
Ex: pe munte și dorești să ajungeți în vf. și este ceață.

Se propun 4 direcții N S E V și se face pasul în direcția care ne duce mai sus

problema e că poate să apară un maxim local



A local max



B. stâlp de telefon (telephone pole)
— totul este plan

- reprezentare a unei hărți prin contururi de înălțime
- toate direcțiile ne duc la vale => concluzia e că suntem pe vârf
- pb. acută în spații multidimensionale

? căutări în capul nostru?

Macbeth, complete, with Start parser, Story Workbench, Genesis system

	BACKTRACKING	USE ENQUEUED LIST	INFORMED
BRITISH MUSEUM	×	×	×
DEPTH FIRST	✓	✓	×
BREADTH FIRST	×	✓	×
HILL CLIMBING	✓	✓	✓
BEAM	×	✓	✓

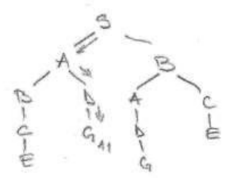
- un sistem monetar cu monede de 1 kron, 7kron, 10kron
  -> cum formezi 15 kron? -> reformularea întrebării / problemei cum formezi 20 kron

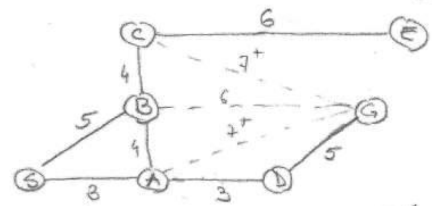
C5 Search: Optimal, Branch and Bound, A*
1/3

- Cum se găsește cea mai bună cale între 2 puncte

- program: poza: care este cea mai scurtă cale între S și G

□ ORACLE
83% □ BRANCH & BOUND
3% + EXTENDED LIST (enqued list) + ADMISSIBLE HEURISTIC
□ A*

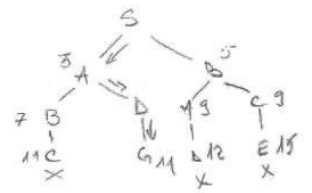




Distanța euristică = distanța dintre 2 noduri în linie imaginară (aeriană)
În general e bine să vă poziționați cât mai aproape de destinație, ținând cont de distanța euristică, dar acest lucru poate produce probleme
Ex. nodul E ar fi o poziție bună pt. că nu e departe euristic de G
→ fundătură

Pe desen, care e cea mai scurtă cale între S și G: SADG 11

Principiul oracolului în rezolvarea unei probleme:
Dacă vrei să rezolvi o problemă, metoda cea mai ușoară e să întrebi pe cineva care cunoaște deja răspunsul
E important să ai încredere în oracol. Dacă nu ai, verifici răspunsul.
Verificarea, în cazul exemplului nostru, este demonstrarea că toate celelalte căi sunt mai lungi decât cea a ORACOLULUI

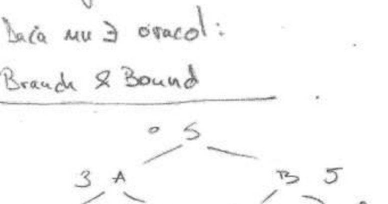


- se extinde cea mai scurtă cale posibilă.
S-B 5
- se extinde calea cea mai scurtă
S-A-B (7) = distanța acumulată până acum
- distanța acumulată se notează în dreptul fiecărui nod
Data trecută s-a discutat despre cât de departe suntem de țintă
De data aceasta ne interesează cât de departe am ajuns

S-A-B-C 11 x - nu se mai extinde pt. că deja s-a ajuns la dist. oracolului
analog SBAD, SBCE

Dacă nu ∃ oracol:

Branch & Bound

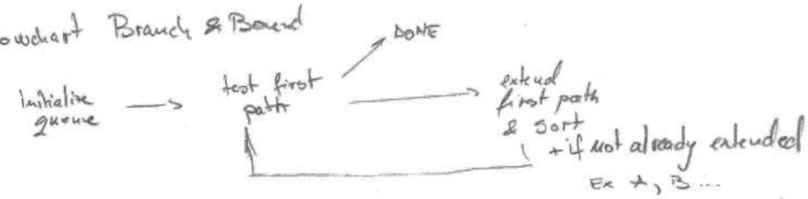


- se extinde calea cea mai scurtă până acum
(S-A); (SAB SAD) SB;
- se pierde ceva din munca pe care am desfășurat-o
Nu. Pt. că e utilă at. când se face verificarea
- după ce am găsit G, se verifică și celelalte căi

Tedlara B&B pe comp.

C5

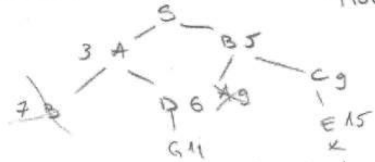
2/3 Flowchart Branch & Bound



B & B + extended list

++ by acc dist + admissible heuristic

Flowchart +



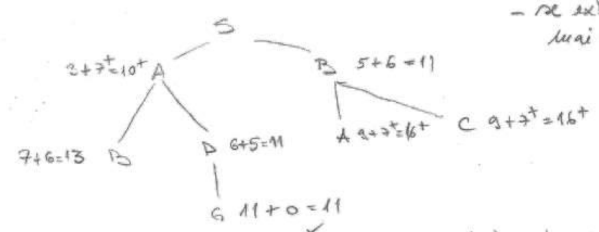
→ simulare

Nu este vorba de un alt algoritm, ci de adăugarea unui strat la cel existent → eficiență crescută
Principiul pistei moarte, o ramură care nu poate fi mai scurtă decât una găsită nu este urmată

B&B + admissible heuristic

se folosește distanța euristică (trebuie să fie mai mică sau egală cu distanța pe graf)

– se extinde ramura cu distanța potențial cea mai scurtă: A



simulare

Caz 1 (st. hărții) Caz 2 (contrul hotelu) (se observă că la B&B + e l. se caută și în stg)

A*

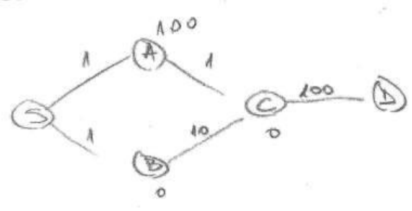
A* = B&B + extended list + admiss. heuristics

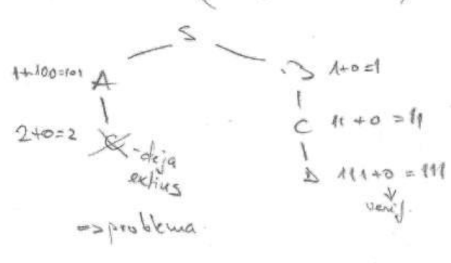
Flowchart ++

Sortarea nu este necesară, trebuie doar ținut cont de cine este minimul

test first path → shortest path

Pt. situații când nu e vorba de hărți – Spații ne Euclidiene; (E.L. – ok, A H – not ok)





→

C5
3/3

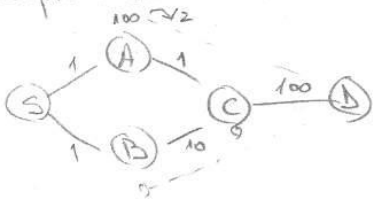
Până acum am folosit o funcție euristică de admitere (admisibile heuristic) dar aceasta nu este suficientă.

A.H: $H(x, Goal) \leq D(x, Goal)$

H - distanța admisibilă D - distanța reală

Consistență $|H(x,G) - H(y,G)| \leq D(x,y)$

În exemplul nostru



A.H: $H(A,D) \leq D(A,D) \Leftrightarrow 100 \leq 1 + 100$ (TRUE)

C.H: $|H(A,D) - H(B,D)| \leq D(A,B) \Leftrightarrow 100 \leq 2$ (FALS)

=> condiția de consistență nu e satisfăcută -> nu funcționează

dacă A estimat 100 -> 2 => succes.

Recapitulare:
- căutare în hărți / grafuri
- căi optime / căi bune

Condiția de
Euristică admisibilă - distanța admisibilă trebuie să fie mai mică decât distanța reală
Euristici consistente - diferența dintre distanța admisibilă de la 2 puncte la țintă trebuie să fie mai mică sau egală cu distanța reală dintre 2 puncte.