

Problem-Solving Agents & Uninformed Search Strategies

Problem-Solving Agents

Een problem-solving agent is een soort goal-based agent.

Het doel help het gedrag van de agent te organiseren door de doelen te limiteren.

Probleem formulering

De probleem formulering bevat de volgende onderdelen:

- Initiële staat
- Mogelijke acties
- Transitie model: wat doet iedere actie
- Doel test
- Path cost: reflectie van de performantie

Probleem oplossing

Een oplossing is sequentie van acties (=pad) dat leidt van de initiële staat naar de doel-staat;

De optimale oplossing is de oplossing met de laagste path cost.

Abstractie

Een goede abstractie verwijdert zo veel mogelijk details als mogelijk zonder dat de geldigheid van de abstractie wegvalt in de echte wereld.

Basic search strategies

Tree-search algoritme

1. De wortel bevat de initiële staat. Hier moet men ook onmiddellijk checken of dit niet het doel is.
2. Expandeer de nodes
3. Kies een blad
4. Kijk of deze de oplossing is
5. Expandeer de node

Hierbij zijn verschillende problemen

1. Loopy paths: Er wordt in een cirkel gegaan
2. Redundant paths: Er zijn meerdere mogelijkheden om van A naar B te gaan

Graph-search algoritme

Vergelijkbaar met tree-search, maar deze houdt een lijst bij met reeds verkende nodes.

Hierdoor kunnen er geen loopy paths meer ontstaan.

Uninformed search strategies

Uninformed search algoritmes hebben geen extra informatie over de staat dan wat gedefinieerd is door de probleem definitie.

De blind search strategieën verschillend dan vaak ook enkel op de volgorde waarin de nodes geëxpandeerd worden.

Breadth-first search

1. Expandeer de root eerst
2. Expandeer de kinderen van de root
3. Expandeer de kinderen van de kinderen van de root
4. ...

Laag per laag uitwerken

Tip: Test de nodes voor ze op de grens stack te zetten.

Diepte eerst geeft een complete oplossing terug als het aantal branches eindig is.

Diepte eerst kan optimaal zijn. Het geeft telkens het minst aantal stappen terug (in node vorm) maar dit is niet altijd gelijk aan de minimale path cost.

Diepte eerst is tijds intensief: $O(b^d)$ met b het aantal gegenereerde nodes op ieder niveau en d de diepte van de oplossing.

Diepte eerst is ook geheugen intensief: Iedere node die gegenereerd wordt blijft in het geheugen:

Explored set: $O(b^{d-1})$

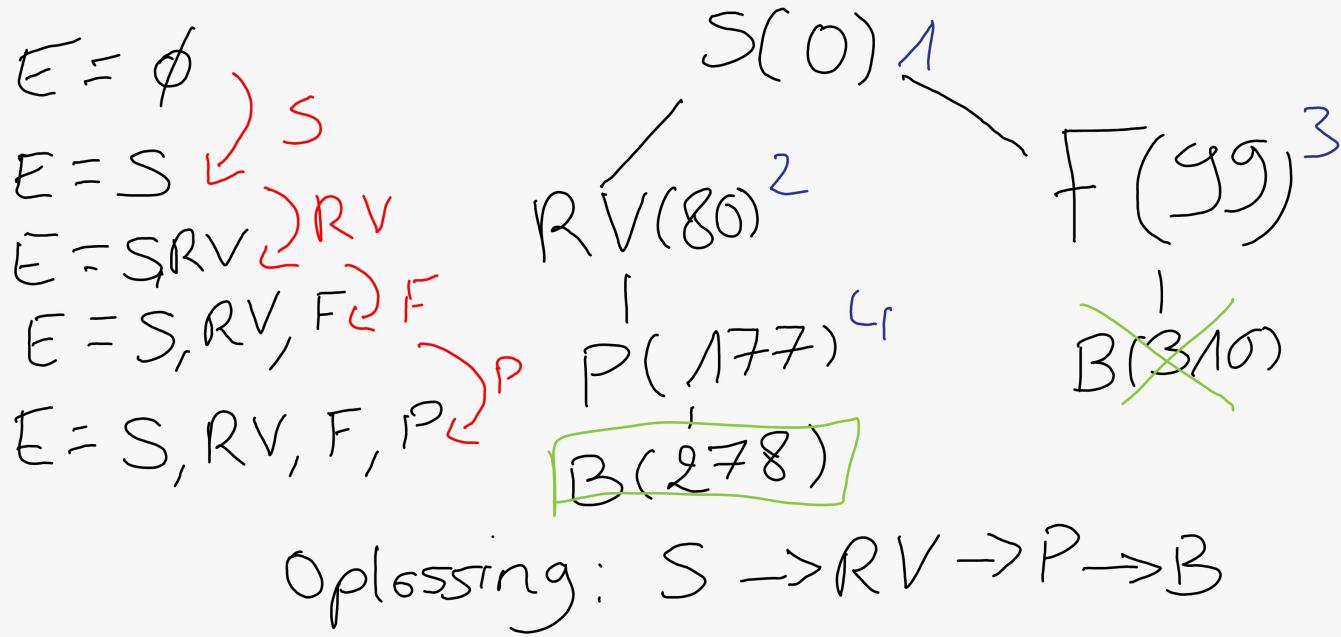
Grens set: $O(b^d)$

De tijd complexiteit is niet goed, maar het gebruikte geheugen is nog veel erger.

Uniform-cost search

cfr. Dijkstra's algoritme

Bij uniform-cost search wordt de grens als priority queue geïmplementeerd



UCS is compleet als de stapkost nooit 0 is, anders kunnen er oneindige lussen optreden.

UCS is optimaal.

Tijd en ruimte kunnen nog erger zijn dan bij BFS. Dit komt doordat eerst de paden met kleine stappen gekozen worden en dan pas deze met grote, misschien nuttige stappen.

Depth-first search

DFS implementeert de grens als LIFO queue

Depth-limited search

DLS is een oplossing voor het gedrag van DFS in de oneindige ruimte.

DLS behandelt nodes op een bepaalde diepte l als nodes die geen kinderen hebben.

Probeer l dus zo te kiezen dan de oplossing zich niet lager dan l bevindt.

bv. Land met 20 steden en men moet de weg zoeken: $l = 19$

Iterative deepening search

DLS iteratief toepassen en telkens l verhogen.

Bidirectional search

Idee: $b^{d/2} + b^{d/2} < b^d$

Start voorwaarts en achterwaards zoeken

Achterwaards zoeken kan enkel als men de voorgangers van een staat kan berekenen.

Vergelijking

Criterium	BFS	UCS	DFS	DLS	IDS	BDS
Compleet	ja	ja	nee	nee	ja	ja
Tijd	$O(b^d)$	$O(b^d)$	$O(b^m)$	$O(b^l)$	$O(b^d)$	$O(b^{d/2})$
Ruimte	$O(b^d)$	$O(b^d)$	$O(bm)$	$O(b^l)$	$O(bd)$	$O(b^{d/2})$
Optimaal	ja	ja	nee	nee	ja	ja